APPENDIX I

ODSI Coalition
August 2000
Version 2.0

N. Ghani
Sorrento Networks
Z. Zhang
Sorrento Networks
L. Zhang
Sorrento Networks
J. Fu
Sorrento Networks
J. Moy
Sycamore Networks

COPS Usage for ODSI

Status of this Memo

This document is a working document of the Optical Domain Service Interconnect (ODSI) Coalition. The goal of the ODSI coalition is to develop technical recommendations for interoperable solutions that allow internetworking devices to dynamically request bandwidth from the optical core, validate the solutions with interoperability testing, and hand off the recommendations to official standards groups.

Copyright Notice

Table of Contents

Abstract

ODSI is a framework of protocols designed to allow internetworking devices to dynamically request bandwidth from optical networks. Within this framework, a protocol is required for policy provisioning inside the optical domain. This documents defines the application of the IETF Common Open Policy Service (COPS) protocol for this purpose and details its interworking with the ODSI framework.

1. Introduction

The ODSI interface will allow intelligent optical networks to interface with client networks and provide advanced services such as bandwidth on demand,

point-and-click provisioning, on-line bandwidth upgrades, etc. In order to support these services, the optical network has to provide appropriate mechanisms that ensure accurate and authorized use of network resources and client accountability. Collectively, these mechanisms are often referred to as policy control.

Policy control refers to a set of administrative criteria used to decide whether or not a service request can be granted to a requesting device. In the case of the ODSI interface, policy control decisions should apply to bandwidth request actions received from the client-side. Policy-based criteria are beyond the regular resource considerations that have to be taken into account in deciding whether an optical lighpath circuit request can be accommodated within the optical network. Policy rules may define conditions on parameters such as source and destination addresses, priorities, bilateral agreements among service providers, time-of-day constraints, cost constraints, etc. It is also generally understood that policy control provides the necessary mechanisms to perform accounting. Upon initial deployment, policy control rules might rely on simple rules, e.g., "approve all requests on behalf of a given user group received from a given ODSI client, if the identity of the requestor can be verified." As more experience is gathered from initial deployments, it is expected that policy rules will become increasingly more and more sophisticated.

The IETF Common Open Policy Service (COPS) protocol is a well-defined policy provisioning protocol. Owing to its generic specification, COPS can also be applied within the optical networks space for authentication and resource control functions. This is a crucial requirement for optical networks and will allow operators to specifically engineer policies to meet their operational requirements. Along these lines, this document describes the application of the COPS protocol within the ODSI framework. A brief introduction to the COPS protocol is first presented, and subsequently, its interworking with the ODSI protocols framework is detailed.

2. Review of the COPS Protocol

A very brief outline of the COPS protocol framework is first given. However, this summary is only intended to serve as a background reference, and interested readers are referred to the various protocol specification documents [Ref1, Ref2,Ref6] for full details.

2.1 Functional Overview

COPS is a query/response protocol based upon a client/server model and has been designed for policy and admission control for a generic set of network resources. Although it is being developed by the IETF RAP (RSVP Admission Policy) group, owing to its inherently generalized design, it can clearly be applied under a much broader context. The framework defines client entities, termed Policy Enforcement Points (PEPs), and server entities, termed Policy Decision Points (PDPs), which exchange policy information through various message types. At least one PDP entity has to be defined for an administrative domain. The messages include request, update, and delete messages from the PEP and decision and update messages from the PDP server (see Section 2.2 also). COPS has been designed to support multiple policy clients [Ref2], and examples include quality of service (QOS), security, customer-specific, etc. Therefore, to distinguish between different client types, a client type must be specified in each message. Each client type can

have its own specific data and policy decision rules. Note that a single PEP or PDP can support policy provisioning for multiple client types.

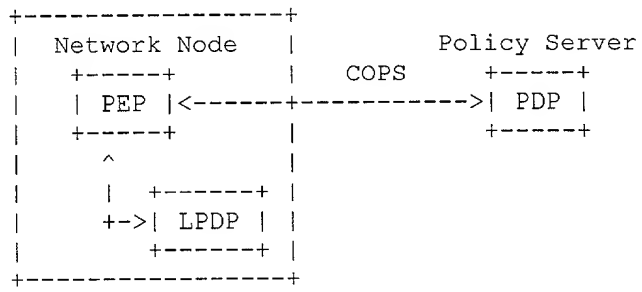


Figure 1: A COPS illustration (from [Ref1]).

The overall interaction between the respective COPS protocol entities is shown in Figure 1 (from [Ref1]). Specifically, the COPS protocol message set details the information exchange between the client PEP and remote PDP server. Additionally, an optional Local Policy Decision Point (LPDP) entity can also be associated with a network device to execute "local" policy decisions. However, all local decision information must be relayed to the PDP also (i.e., by the PEP in the form of an LPDP decision object), and the PDP entity remains as the binding decision point for all PEP requests. A PEP client communicates with its PDP server to implement various policy decision frameworks. As a network client, the PEP initiates this communication by opening a persistent (two-way) TCP connection to the PDP, and this connection is used for all communications between these two entities. TCP is chosen due to the important nature of the policy control information carried. This choice precludes the need for any additional mechanisms for reliable communication.

COPS can function in two basic models, outsourcing and provisioning [Ref2] and defines various objects for its functioning, see [Ref1]. In the outsourcing model represents a client-driven approach, where the PDP server actively responds to incoming policy requests from PEP clients. This model works well for signaled protocols, e.g., as in RSVP-COPS [Ref6]. Client requests are carried via COPS-PR defined data objects, which communicate Client Specific Information objects [Ref7]. The format of this data is independent of the actual messaging protocol, and hence this decouples the information model from the actual signaling semantics. For example, RSVP-COPS requires all RSVP Path message data to be directly encapsulated into associated COPS message types (see [Ref6] for details). Examples of various RSVP PEP-PDP message exchange sequences are also detailed in [Ref6].

Meanwhile, the provisioning model represents a server-driven approach, where the PDP downloads (pre-provisions) policy information to client PEPs beforehand. Such policy information is based upon predicted configurations/demands and works well for non-signaled protocols/scenarios. Specifically, after the PEP-PDP connection is setup, the PEP sends a configuration request message to the PDP, which in turn replies with a message containing all provisionable policies for the PEP device. Alternatively, the PDP can also asynchronously "push" policy state onto the PEP. Meanwhile, the actual COPS policy data is represented via a named "self-identifying" data structure, termed the Policy Information Base (PIB) [Ref2]. This structure specifies the type and purpose of the policy information arriving from the PDP, and hence must also be client specific.

This information is installed by the PEP. Conceptually, PIB can be described using a tree structure, consisting of Policy Rule Classes (PRCs) and their associated Policy Rule Instances (PRIs), see [Ref2]. The overall PIB concept is very flexible, and the class/rule definitions can be further modified between PEP/PDP nodes to reflect new policy requirements, e.g., expiration quotas, time-of-day restrictions, and other SLA details (see [Ref2]). In the provisioning model, the PDP can re-issue updated policies to PEPs at any time, i.e., asynchronous updates. After any installation/deletion of such configuration data from the PDP, PEP nodes must send appropriate report messages to the PDP for accounting and monitoring purposes.

COPS relies upon a state-based model, where the request/decision state is shared between the PEP and PDP. Specifically, PEP requests are stored on the PDP until explicitly deleted (signaled) from the PEP. These stored requests can have an impact on how the PDP servers process future requests (i.e., based upon current request/decision states). PEP clients whose request states change must notify their PDP servers to the effect and delete non-applicable state information. However, PDP servers can also issue unsolicited messages (e.g., state updates) to PEP clients in order to force existing states. This may be necessary during policy upgrades or SLA changes. Depending upon the client type, multiple states can be installed for a single PEP/PDP relationship (as identified by the Handle object).

Due to the important role of the COPS protocol, fault-tolerance features are also provided. These capabilities allow to "re-synchronize" state between the PEP and PDP in the event of a communications failure. The PEP-PDP connection is constantly verified using keep-alive messages, and upon failure detection, the PEP falls back to local decisions (via an LPDP). While disconnected from the PDP, the PEP tries to reconnect to the original PDP and/or to an alternative PDP [Ref1]. Upon connection re-establishment, the PEP must provide the PDP with new state and/or event information. Furthermore, the PDP can resynchronize all the PEP's internal state. See [Ref2] also for additional details on fault tolerance. Finally, security provisions for COPS are provided via an Integrity object, and all COPS implementations must support this object. Additionally, client (PEP) and server (PDP) entities can also use IP Security [Ref8] mechanisms to authenticate and secure the communications channel between the PEP and the PDP entities, as described in [Ref1].

2.2 Message Set Summary

A very brief summary of the COPS message set is presented to give a high-level view of some of the protocol's internals. Interested readers are referred to [Ref1],[Ref2] for more details:

o Request (REQ) (PEP-to-PDP): PEPs send REQ messages to request policy provisioning data from the PDP. This message includes client-specific information to assist the PDP and also a unique Client Handle (to identify request states at the PEP and PDP entities).

o Decision (DEC) (PDP-to-PEP): PDPs respond to client REQ messages via DEC messages. These messages contain multiple policy decisions that are to be installed on the PEP. DEC messages can be used to delete/update existing rules (i.e., state) in the PEP. DEC messages use the Client Handle to identify the REQ being responded to. These messages contain command codes which instruct various operations on client-specific PIB entities.

o Report State (RPT) (PEP-to-PDP): This message is sent from the PEP to the PDP to report accounting information. Also, a PEP must always report back to the PDP the outcome of action taken for an earlier DEC (install) message, i.e., indicating success or failure in applying the configuration action [Ref1].

o Delete Request State (DRQ) (PEP-to-PDP): This message is sent to the PDP in order to delete a request (or related information) pertaining to the specified Client Handle. PEPs can issue DRQ messages in response to status changes on their end (e.g., connection requests withdrawn by clients), and the PDP will take appropriate actions to remove state. It is important for PEPs to issue this command to ensure state consistency with the PDP.

o Synchronize State Request (SSQ) (PDP-to-PEP): This message requests the PEP to re-send state information. The Client Handle field here is optional, and if specified, only the respective state needs be conveyed (via a PEP RPT message). If, however, no Client Handle is specified, the PEP must send all its state information to the PDP.

o Client-Open (OPN) (PEP-to-PDP): This message is sent to the PDP server to specify various PEP-related information: client-types supported, last PDP connected to per client type, etc. Multiple such messages can be sent at anytime. PEP clients normally send an OPN message after connection establishment with a PDP server.

o Client-Accept (CAT) (PDP-to-PEP): In response to the PEP OPN message, the PDP can respond "positively" with a CAT message. This reply contains various timer values which are used to generate keep-alive and/or accounting report messages.

o Client-Close (CC) (PEP-to-PDP, PDP-to-PEP): This is a bi-directional message and indicates that a particular client type is no longer supported. This can be sent from the PDP in response to a PEP OPN message, in which case an alternative PDP server may also be specified.

o Keep-Alive (KA) (PEP-to-PDP, PDP-to-PEP): These messages are transmitted by the PEP in order to maintain the PEP-PDP connection (i.e., independent of any particular client-type). Associated timer values for this message are specified by the PDP (e.g., via CAT response). The PDP must echo a keep-alive ACK message per incoming PEP keep-alive message.

o Synchronize State Complete (SSC) (PEP-to-PDP): This message is sent by the PEP after receiving a PDP SSQ message, and indicates that (state) synchronization between the PEP and PDP is complete. This message may or may not include a Client Handle (as per the original PDP SSQ request message).

3. COPS Applied to the ODSI Framework

Clearly, COPS provides a very generic policy control framework that is very extensible to the optical networking domain. With recent trends towards IP-based control for optical networks, the use of this framework will further provide for a more seamless interworking with IP-controlled devices. Moreover, the built-in reliability and security features of this protocol ensure its applicability to robust services-provisioning scenarios. Hence, it is logical to interwork the COPS and ODSI frameworks together in order to provide a comprehensive policy provisioning setup for optical networks and

extend optical policy provisioning closer to the network edge. In particular, the ODSI interface on edge optical devices must communicate with a COPS PEP entity in order to relay policy provisioning request/responses messages. Since the PEP deals with optical network policy administration, it must reside in an edge optical device. This overall framework is illustrated more clearly in Figure 2, where the PEP client communicates with the ODSI interface.

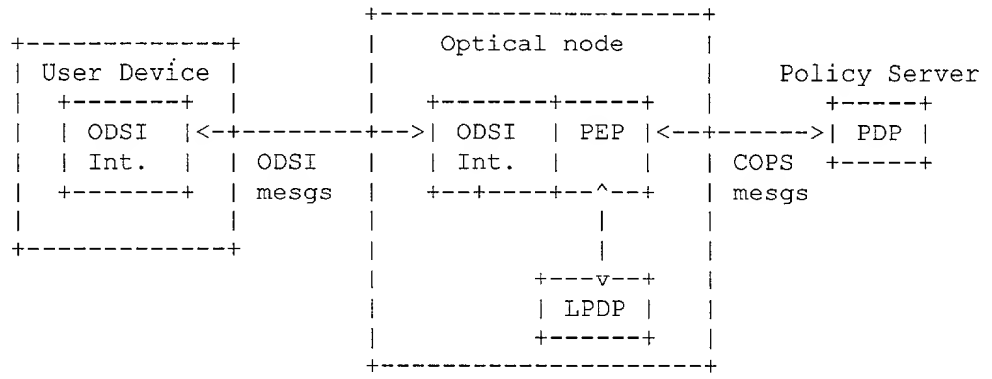


Figure 2: COPS applied to the ODSI framework

Initially, the ODSI user device-to-optical interface is setup by a series of procedures (service discovery, address registration, signaling, see [Ref3],[Ref4],[Ref5]). After this procedure is complete, the COPS startup procedures can be initiated. In the above interworking, the ODSI interface residing in an optical device passes messaging between the (optical domain) PEP and the (electronic domain) user device, e.g., such as an IP router or SONET digital cross-connect. In particular, this entity translates ODSI actions into appropriate PEP client actions. Subsequently, the optical PEP entity sends back appropriate reply messages to the ODSI interface. Here, the outsourcing policy management model is applied for COPS-ODSI interworking (akin to RSVP-COPS interworking [Ref6]), and ODSI actions are transmitted to the PDP via the local PEP entities. In this case, the PDP (LPDP) returns policy responses to the PEP, which are in turn translated into appropriate ODSI interface actions and/or messages. Therefore, when PEP's open communications with PDP servers, their initial Client-Open (OPN) messages have the client-type set to indicate a signaled (ODSI) client. If the PDP supports this client-type, it responds with a Client-Accept (CAT) message or otherwise with a Client-Close (CC) message. Alternatively, the PDP can also redirect the PEP to other PDPs in the network domain (via a Redirect address in the CC message). After receiving a CAT message, the PEP can issue the first (ODSI-driven) request message to the PDP server.

Meanwhile, the information transfer model assumes that all objects received in the ODSI messages are encapsulated in a Client Specific Information Object (Signaled ClientSI) and sent from the PEP to the PDP. Specifically, ODSI trail request messages will contain topological information (source, destination IP addresses and port numbers), bandwidth requirements, protection levels, preemption priorities, and setup priority. It assumed that the (ODSI policy server) PDP is fully ODSI-aware and can handle these message contents and generate appropriate policy control decisions.

ODSI clients are divided into user groups, and bandwidth creation is strictly restricted to (client) user devices belonging to the same OSDI user group [Ref5]. In particular, "admission control" is applied based upon the user group and requester and endpoints (as per functional specification [Ref1]). Hence, when fielding bandwidth requests, ODSI components must first resolve the IP addresses of the endpoints and resolve the associated user group value. This information, along with the details of the ODSI bandwidth request is then forwarded to the COPS policy server (i.e., optical network PDP), which will perform the necessary policy provisioning and accounting procedures on the request. Also note that the ODSI specification states that (electronic client) endpoints may be partially specified [Ref3]. By this it is meant that the complete endpoint information may not be given (i.e., port numbers). However, the endpoint IP addresses are always specified, and user group identification should be possible with this information alone.

As per the ODSI signaling specification [Ref4], the request is propagated from (trail) requester to (trail) head to (trail) tail, and then to an optical network controller (ONC) entity. The ONC validates the request and subsequently allocates the resources for the circuit. Now clearly, one of these ODSI entities must initiate policy provisioning via COPS. Here, it is felt that such policy requests are best sourced by the ONC entity, i.e., PEP entity in Figure 2. Since this network entity is intended to validate the (ODSI) request, this placement is entirely logical. If the policy requirements are valid, the request can be further processed, otherwise, the request must be rejected (by sending an ODSI trail notification message to the trail requester). In the case the ONC device does not support ODSI-COPS interworking (i.e., no PEP entity), the request must again be rejected. In the following section, the message formats will be specified in more detail.

4. Message Contents

The COPS protocol provides the capability for different COPS clients to define their own "named", i.e. client-specific, information for various messages. This section describes the messages exchanged between a COPS server (PDP) and COPS ODSI clients (PEP) that carry client-specific data objects. Since ODSI is a signaled client, the client-type is always set to 5 (type ODSI) in the common header field.

4.1. Request (REQ) PEP -> PDP

The REQ message is sent by COPS-ODSI clients in response to various bandwidth request actions. The actions can comprise requesting a new trail, modifying or non-destructively modifying a previously established trail, querying a trail, and releasing a trail. The Client Handle associated with the REQ message originated by a client must be unique for that client. The Client Specific info object is used to specify the requested resources.

Each REQ message contains a single request. The PDP responds to the resource allocation request with a DEC message containing the answer to the query. The REQ message has the following format:

```
<Request> ::= <Common Header>
              <Client Handle>
              <Context >
              [<ClientSI: all objects in ODSI Trail Create Request >]
              [<LPDPDecision(s)>]
              [<Integrity>]
```

Only those ODSI "request" messages that are sourced by the trail requester are supported by COPS, namely Trail Create Request, Trail Modify Request, Trail Query, and Trail Destroy Request. The other ODSI message types are not supported by COPS.

All objects that are received in the respective (requester-sourced) ODSI messages are encapsulated inside the Client Specific Information (ClientSI) Object sent from PEP to the remote PDP.

<ClientSI: all objects in Trail Create Request>, or
<ClientSI: all objects in Trail Modify Request>, or
<ClientSI: all objects in Trail Query>, or
<ClientSI: all objects in Trail Destroy Request>.

If the ODSI message is a Trail Modify Request message, then the previous value for starting time should also be appended at the end of ClientSI object (for accounting purpose):

```
<ClientSI: all objects in Trail Modify Request>,
                        <starting time>      (new value)
                        <starting time>      (old value)
```

The LPDPDecision can be specified if applicable.

4.2. Decision (DEC) PDP -> PEP

The DEC message is sent from the PDP to a COPS-ODSI client in response to the REQ message received from the PEP. Unsolicited DEC messages cannot be sent for this client type (therefore the solicited decision flag is always set). The Client Handle must be the same Handle that was received in the REQ message.

PDP performs admission control for ODSI message based on the User Group Id, and some other criteria. Each DEC message contains a single decision. The DEC message contains decision objects or error objects [Ref1]. For the COPS-ODSI client type, the DEC message has the following format:

```
<Decision Message> ::= <Common Header>
                       <Client Handle>
                       <Decision> | <Error>
                       [<Integrity>]
```

The decision object in turn has the following format:

```
<Decision> ::= <Context>
               <Decision: Command code>
               <Decision: Client SI data>
```

The context object will be the same as contained in the REQ message. The <Decision: Command code> object will contain the answer in the Command-code field according to the COPS specifications. In particular the Command-code will be "Install" to mean a positive answer and "Remove" to mean a negative answer. The following text clarifies how Install and Remove Decisions map into the different (supported) ODSI message types.

REQ (M-Type= Add)

DEC (Install) -> Requested resources are successfully allocated

DEC (Remove) -> Requested resources are not allocated

REQ (M-Type = Release)

DEC (Install) -> Resources are released

DEC (Remove) -> Resources are still allocated.

REQ (M-Type= Modify)

DEC (Install) -> Modification is accepted. The newly requested resources are allocated, while the previous ones have been released

DEC (Remove) -> Modification is not accepted. Previous allocation is still active.

REQ (M-Type=Query)

DEC (Install) -> Request for query is accepted.

DEC (Remove) -> Request for query is not accepted.

The Error object is used to communicate COPS protocols error to the PEP, according to the definition in [Ref1]. Some examples include bad handles, bad/malformed message, unsupported client type, mandatory object missing (in REQ message), communications failure, shutting down, authentication issues (see [Ref1] for further details and associated codes). No (ODSI) client-specific error sub-codes are defined for COPS-ODSI interworking.

The Decision ClientSI data object, meanwhile, carries the ODSI-related information needed to correlate the decision with the answer and some optional information to explain negative Decisions. It has the following format:

<Decision: Client SI data> ::= <Reject Reason Code>

Possible reasons are specified as follows:

Reject Reason Code (1 byte):

1=Resource unavailable
2=Unsupported resource type
3=Unacceptable source address
4=Unacceptable destination address

The PEP must report the success or failure in applying the (configuration) in the decision message via a report message [Ref1]. This allows the PDP to maintain correct state and also for the PEP to report specific errors.

4.3. Report State (RPT) PEP -> PDP

COPS PEP's generate RPT messages for the PDP server. These messages are required to report the outcome of the PDP's DEC messages via a RPT message,

with the solicited flag set. The Report-Type field specifies the outcome of the report (i.e., success or failure). On reception of Report State messages (especially those with Report-Type failure), the PDP can choose to start a Synchronization procedure. The RPT message for the COPS-ODSI client type has the following format:

```
<Report State Message> ::= <Common Header>
                           <Client Handle>
                           <Report Type>
                           [<Integrity>]
```

4.4. Synchronize State Request (SSQ) PDP -> PEP

The Synchronize State Request message is sent by the PDP to the PEP to "reset" the state information. It requests the PEP to send the set of resource allocation REQ messages needed to rebuild the state. The SSQ can apply to the whole set of PEP active reservations PEP, or to a specific resource type and source-destination couple, depending on the information contained in the Client SI object.

```
< Synchronize State> ::= <Common Header>
                         <Client Handle>
                         <ClientSI: SSQ scope>]
                         [<Integrity>]
```

4.5. Synchronize State Complete (SSC) PEP -> PDP

The Synchronize State Complete message is sent by the PEP to the PDP to inform that all the REQ messages needed to rebuild the state have been sent. The Client SI object is the same received in the SSQ message and specifies the scope of the synchronization procedure which has been completed.

```
< Synchronize State Complete> ::= <Common Header>
                                  <Client Handle>
                                  <ClientSI: SSQ scope>]
                                  [<Integrity>]
```

5. Illustrative Examples

In this section, some illustrative examples for the COPS-ODSI client interworking are presented. In the first example, the trail request is successful, and in the second example the request is rejected.

A PEP requests an OC48 trail between Head (192.234.124.1) and Tail (192.234.4.1).

```
PEP -- > PDP   REQ: = <Handle  C>
                   <Context: Trail Create Request>
                   <ClientSI:
                            ** begin of ODSI Trail Create Request message **
                           User Group:          X
                           Source Address:      192.234.124.1
                           Destination address: 192.234.4.1
                           Bandwidth type:      OC48
                            ** end of ODSI Create message  ***
                  >
```

The PDP accepts the request.

```
PDP--- > PEP     DEC: = <Handle  C>
                       <Context: Trail Create Request>
                       <Decision: Command = Install>
                       <Decision: Client SI· >
```

In the following example, the trail request of an OC192 between Head (192.234.1244.1) and Tail (192.234.4.1) is rejected.

```
PEP -- > PDP         REQ: = <Handle  C>
                         <Context: Trail Create Request>
                         <ClientSI:
                             ** begin of ODSI Trail Create Request message **
                           User Group:          X
                           Source Address:      192.234.124.1
                           Destination address: 192.234.4.1
                           Bandwidth type:      OC192
                             ** end of ODSI Create message  ***
                         >

PDP--- > PEP     DEC: = <Handle  C>
                       <Context: Trail Create Request>
                       <Decision: Command = Remove>
                       <Decision: Client SI
                            Reason code: resource unavailable >
```

5. References

[Ref1] Durham, D., Boyle, J., R. Cohen, S. Herzog, R. Rajan and A. Sastry, "The COPS (Common Open Policy Service) Protocol", IETF RFC 2748, January 2000.

[Ref2] Chan, K., Durham, D., Gai, S., Herzog, S., McCloghrie, K., Reichmeyer, F., Seligson, J., Smith, A., Yavatkar, R., "COPS Usage for Policy Provisioning", IETF Draft draft-ietf-rap-pr-02.txt, March 10, 2000.

[Ref3] ODSI Coalition, G. Bernstein, R. Coltun, J. Moy and A. Sodder, editors, "Optical Domain Service Interconnect (ODSI) Functional Specification", March 2000.

[Ref4] Bernstein, G., Coulton, R., Moy, J., Sodder, A., "Optical Domain Service Interconnect (ODSI) Signaling Specification", April 2000.

[Ref5] Bernstein, G., Coulton, R., Moy, J., Sodder, A., Arvind, K., "ODSI Service Discovery and Address Registration," April 2000.

[Ref6] Herzog, S., Boyle, J., Cohen, R., Durham, D., Rajan, R., Sastry, A., "COPS Usage for RSVP," IETF RFC 2749, January 2000.

[Ref7] Durham, D., Khosravi, H., Weiss, W., Avri, D., "Cops Usage for AAA," IETF Draft draft-durham-aaa-cops-ext-00.txt, May 2000.

[Ref8] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 2401, August 1995.

Editors' Addresses

Nasir Ghani
Sorrento Networks Inc.,
9990 Mesa Rim Road
San Diego, CA 92121
Phone: (858) 646-7192
Email: nghani@sorrentonet.com

Zhensheng Zhang
Sorrento Networks Inc.,
9990 Mesa Rim Road
San Diego, CA 92121
Phone: (858) 646-7195
Email: zzhang@sorrentonet.com

Leah Zhang
Sorrento Networks Inc.,
9990 Mesa Rim Road
San Diego, CA 92121
Phone: (858) 450-4977
Email: leahz@sorrentonet.com

James Fu
Sorrento Networks Inc.,
9990 Mesa Rim Road
San Diego, CA 92121
Phone: (858) 450-4924
Email: jfu@sorrentonet.com

John T. Moy,
Sycamore Networks,
10 Elizabeth Drive,
Chelmsford, MA 01824
Phone: (978) 367-2161
Email: jmoy@sycamorenet.com

APPENDIX II

ODSI Coalition
October 9, 2000
Version 1.5

K. Arvind
Tenor Networks
R. Coltun
Siara Systems
J. Moy
Sycamore Networks
A. Sodder
Tenor Networks
J. Weiss
Ciena
Editors

Definition of Managed Objects for ODSI Management

Status of this Memo

This document is a working document of the Optical Domain Service Interconnect (ODSI) Coalition. The goal of the ODSI coalition is to develop technical recommendations for interoperable solutions that allow user devices to dynamically request bandwith from the optical core, validate the solutions with interoperability testing, and hand off the recommendations to official standards groups.

Copyright Notice

Table of Contents

1. Abstract

This memo defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community. In particular, it describes objects for managing the

services and protocols defined by the Optical Domain Service Interconnect (ODSI) Coalition, that enable user devices to dynamically request bandwidth from the optical core.

Textual Conventions used in this MIB are defined in STD 58, RFC 2579 [RFC2579].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. The SNMP Network Management Framework

The SNMP Management Framework presently consists of five major components:

o An overall architecture as defined in RFC 2571 [RFC2571].

o Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16, RFC 1155 [RFC1155], STD 16, RFC 1212 [RFC1212] and RFC 1215 [RFC1215]. The second version, called SMIv2, is described in STD 58, RFC 2578 [RFC2578], STD 58, RFC 2579 [RFC2579], and STD 58, RFC 2580 [RFC2580].

o Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and is described in STD 15, RFC 1157 [RFC1157]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [RFC1901], RFC 1905 [RFC1905], and RFC 1906 [RFC1906]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [RFC1906], RFC 2572 [RFC2572], and RFC 2574 [RFC2574].

o Protocol Operations for accessing management operations. The first set of protocol operations and associated PDU foprmats is described in STD 15, RFC 1157 [RFC1157]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [RFC1905].

o A set of fundamental applications described in RFC 2573 [RFC2573] and the view-based access control mechanism described in RFC 2575 [RFC2575].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [RFC2570].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

This memo specifies a MIB module that is compliant to SMIv2. A MIB conforming to SMIv1 can be produced through the appropriate translations. The resulting translated MIB must be semantically equivalent, except where objects or events are omitted because no translation is possible (e.g., use of Counter64). Some machine-readable information in SMIv2 will be converted into textual descriptions in SMIv1 during the translation process. However, this loss of machine readable information is not considered to change the semantics of the MIB.

3. ODSI Overview and Terminology

This section provides an overview of the services and protocols defined by ODSI, and identifies the key entities involved. The purpose of this overview is to provide a context for understanding the structure of the ODSI MIB. A detailed description of ODSI may be found in the ODSI Functional Specification [ODSIFS] and individual ODSI protocol and service specification documents.

The Optical Domain Service Interconnect (ODSI) specification defines a set of protocols and services, that constitute an optical user-to-network interface (optical UNI), an interface between a client of the optical core and the optical core itself. The purpose of this UNI is to enable user devices (client internetworking devices such as routers, packet switches, ATM switches, SONET ADMs) to dynamically request bandwidth from the optical network core in an on-demand fashion.

The service provided by the ODSI UNI is the dynamic creation and management of point-to-point bandwidth channels between user devices attached to the optical network. The ODSI UNI is based on the Internet protocol suite and IP addressing, and defines extensions to existing protocols such as PPP, RSVP, and LDP.

ODSI assumes the following model. A user device is connected to the optical network through one or more physical interfaces, such as OC-48 links. The user device wishes to dynamically connect these interfaces to other user devices attached to the optical network. The ODSI UNI facilitates the establishment of such a connection on an as-needed basis. When such a connection is established, it looks like a leased line connecting the two user devices. If the user device's interface supports channelization, parts of the interface may be connected to separate destinations; for example, each of the 4 OC-12s in an OC-48 interface may be connected by ODSI to separate remote routers. The ODSI protocols used to signal dynamic bandwidth requests may use either an out-of-band or in-band control channel. An example of an out-of-band control channel is an Ethernet network that connects the user device to the optical network. An in-band control channel specified by ODSI consists of SONET overhead bytes in the SONET interfaces used to connect the user device and network device.

The core entities in the ODSI model are the following:

o User Device: An internetworking device such as a router, packet switch, ATM switch, or SONET ADM, that is a client of the optical network with respect to bandwidth services.

o Network Device: A device such as an optical switch or optical crossconnect that forms part of the optical core. A User Device connects to the optical network by attaching to a Network Device at the boundary of the optical network.

o Port: Ports represent physical points of attachment between user and network devices. A port is just a physical interface (e.g., SONET OC-12 interface) on a user or network device, and is identified by an RFC 2233 ifIndex [RFC2233]. A port is associated with an IP address that serves both to identify the port, and as a source or destination address of ODSI protocol packets. The same IP address may be associated with more than one port in a device, in which case the ifIndex is also required to identify a port. ODSI also allows a network device port to have multiple IP addresses associated with it, for potential use in NNI type applications [ODSIFS].

Note that the IP addresses referred to here are IP address that are routable in the context of the optical network's control network.

o Channel: Some ports, depending on the type of physical interface, may support channelization. For example, a SONET OC-12 interface supports 4 OC-3 channels. On multiplexed interfaces such as this, that allow multiple channels, a user device may request sub-rate bandwidth connections by specifying a particular channel. For example, on a SONET interface, a channel is specified as a list of one or more (for concatenated channels) timeslots.

o Container: This parameter further subdivides a channel. As an example, when the port uses SONET framing, the "container" parameter identifies a virtual tributary within a channel.

o User Group: The concept of an ODSI User Group is analogous to the concept of a Closed User Group in X.25. ODSI connections may be established only between members of the same User Group. For example, if there were 10 ISPs whose routers were all attached to the same optical network, each ISP would have a separate User Group identifier. Each IP address of a user device is associated with a ODSI User Group. The user group is encoded as a VPN Idenitifer [RFC2685].

o End Point: ODSI bandwidth channels (connections) are established between End Points on two user devices. An end point is associated with a specific port on a user device, and a specific channel on the port (if the port supports channelization), and a specific container on the channel (if the channel supports containers). An End Point is automatically associated with the IP address of the port it belongs to.

o Connection: An ODSI Connection is a bandwidth channel established between 2 ODSI End Points. An ODSI connection is signaled either by a device associated with one of the end points, or by a third party device such as a network management station.

An ODSI connection is defined by a number of parameters that together characterize it:

o End Points: The two end points of the connection

o Framing: The format of the signal (e.g., PDH, SONET, SDH, Ethernet, etc.) on the port between user and network devices at each end point. The framing at each end point may be different.

o Transparency: The portion of the signal that is available for user data traffic. For SONET/SDH framing, the transparency options are:

- PLR-C: Physical layer regeneration, meaning all the SONET signal is available for user data.

- STE-C: End points are attached to section-terminating equipment. Only line and path overhead are available to user data traffic.

- LTE-C: End points are attached to line-terminating equipment. Only path overhead is available to user data traffic.

o Size: The amount of bandwidth requested in bits per second. Examples include STS-x, OC-x, 1 or 10 Mbps Ethernet. Any

size bandwidth may be requested, although it is not required that all sizes be supported by the optical network. When a size is requested that the optical network does not support, it may return a larger bandwidth connection, but never a smaller.

o Service Level: Service level encompasses preemption priority, protection, and availability. It is encoded as an integer, whose meaning is assigned by the optical network provider.

o Propagation delay: The maximum amount of propagation delay, specified in milliseconds, to be tolerated on the connection.

o Diversity: The set of ODSI connections with which this connection is required to not share any resources. The type of non-sharing (diversity) may be one of link diverity, node diversity, or SRLG (Shared Risk Link Group) diversity.

o Directionality: An ODSI connection may be either unidirectional or bidirectional.

o Contract ID: This parameter is assigned by the optical service provider and provides billing and authorization information for the ODSI connection.

o Control Channel: An in-band or out-of-band channel between a User Device and the Network Device to which it is attached, that is used to convey signaling protocol requests. A control channel is identified by an RFC2233 ifIndex [RFC2233].

o In-Band Control Channel: An ODSI Port may support an in-band control channel that is multiplexed on the physical channel connecting the port to its ODSI partner (User or Network Device Port). An example of an in-band channel would be a channel consisiting of 1 or more specified SONET overhead bytes.

Availability of an in-band control channel facilitates the automatic configuration of User Device ports using the ODSI Service Discovery and Address Registration Protocols.

The protocols that constitute the ODSI UNI are the following:

o Service Discovery: The ODSI Service Discovery Protocol [ODSIPPP] is used by a User Device Port to determine whether the Network Device Port to which it connects supports ODSI signaling, and for exchanging identification information such as port index, user group, support for channelization, etc.

The ODSI Service Discovery protocol uses the PPP protocol [RFC1661] over the in-band control channel of a port. If the in-band control channel is not available, Service Discovery has to be implemented using manual configuration.

o Address Registration: The ODSI Address Registration Protocol is used by a User Device Port to advertise its IP addresses to the Network Device Port to which it is attached. It may also be used for advertisement of IP addresses between Network Device ports in NNI-type applications [ODSIFS, ODSIPPP].

The ODSI Address Registration protocol [ODSIPPP] uses the PPP protocol [RFC1661, RFC1332] over the in-band control channel of a port. If the in-band control channel is not available, Address Registration has to

be implemented using manual configuration.

o Signaling: The ODSI Signaling Protocol is used to request a connection between two end points, tear down a connection, query the status of a connection, modify certain attributes of an existing connection, or do a directory lookup of available end points. The Signaling Protocol is a TCP application [ODSISIG].

4. Textual Conventions

When designing a MIB module, it is often useful to define new types similar to those defined in the SMI. In comparison to a type defined in the SMI, each of these new types has a different name, a similar syntax, but a more precise semantics. These newly defined types are termed textual conventions, and are used for the convenience of humans reading the MIB module. This is done through Textual Conventions as defined in RFC 2579 [RFC2579].

The ODSI MIB defines and uses the following textual conventions:

o PositiveInteger: A 32-bit positive integer.

o OdsiUserGroup: ODSI User Group.

o OdsiDeviceType: Type of device, User Device or Network Device.

o OdsiConnState: State of an ODSI connection.

o OdsiConnFraming: Framing used with ODSI connection.

o OdsiConnTransparency: Transparency associated with ODSI connection.

o OdsiChannelSpec: Channel specification.

o OdsiContainerSpec: Identfies a container on a channel.

o OdsiInBandChanSonetSpec: SONET in-band channel specification.

o OdsiInBandChanDigitalWrapperSpec: Digital wrapper in-band channel specification.

5. Structure of the MIB

5.1 Scope

This MIB is intended to provide management of ODSI-related functions implemented by User and Network Devices. The scope of this MIB is restricted to the configuration and monitoring of the various entities involved in implementing the ODSI UNI. This MIB does not provide the ability to trigger the set up ODSI connections via SNMP. Such a function is deferred to proprietary applications.

5.2 Object Selection Criteria

Good engineering practice and IAB directives require that a MIB be as simple as possible. Simplicity is realized by applying the following criteria to objects proposed for inclusion:

(1) Require objects be essential for operational management. In particular, objects for which the sole purpose is to debug

implementations are excluded from the MIB.

(2) Consider evidence of current use and/or utility.

(3) Limit the total number of objects

(4) Exclude objects which are simply derivable from this or other MIBs.

5.3 Pictorial Overview of Namespace

```
odsi
   (1)odsiModules
         (1)odsiMibModule
   (2)odsiObjectIdentities
         (1)odsiProtocolVersions
               (1)odsiProtocolVersion0100
   (3)odsiObjects
         (1)odsiDeviceObjects
               (1)odsiDeviceProtocolVersion
               (2)odsiDeviceType
         (2)odsiPortObjects
               (1)odsiPortCount
               (2)odsiPortTable
                     (1)odsiPortEntry
                           (1)odsiPortAdminStatus
                           (2)odsiPortOperStatus
                           (3)odsiPortNumChannels
                           (4)odsiPortLocalAddrCount
                           (5)odsiPortRemoteAddrCount
                           (6)odsiPortSigAddr
                           (7)odsiPortAuthAlgorithm
                           (8)odsiPortLocalPortId
                           (9)odsiPortRemotePortId
                           (10)odsiPortLocalMinChanSize
                           (11)odsiPortRemoteMinChanSize
                           (12)odsiPortLocalConcatenation
                           (13)odsiPortRemoteConcatenation
               (3)odsiPortStatsTable
                     (1)odsiPortStatsEntry
                           (1)odsiPortStatsInCalls
                           (2)odsiPortStatsInCallRefusals
                           (3)odsiPortStatsInCallResets
                           (4)odsiPortStatsInConnections
                           (5)odsiPortStatsOutCallAttempts
                           (6)odsiPortStatsOutCallFailures
                           (7)odsiPortStatsOutCallResets
                           (8)odsiPortStatsOutConnections
               (4)odsiPortLocAddrNewIndex
               (5)odsiPortLocAddrTable
                     (1)odsiPortLocAddrEntry
                           (1)odsiPortLocAddrIndex
                           (2)odsiPortLocAddrIpAddr
                           (3)odsiPortLocAddrUserGrp
                           (4)odsiPortLocAddrStatus
               (6)odsiPortRemAddrNewIndex
               (7)odsiPortRemAddrTable
                     (1)odsiPortRemAddrEntry
                           (1)odsiPortRemAddrIndex
                           (2)odsiPortRemAddrIpAddr
                           (3)odsiPortRemAddrUserGrp
                           (4)odsiPortRemAddrStatus
         (3)odsiInBandChanObjects
               (1)odsiInBandChanTable
```

```
                (1)odsiInBandChanEntry
                      (1)odsiInBandChanSupported
                      (2)odsiInBandChanAdminStatus
                      (3)odsiInBandChanOperStatus
                      (4)odsiInBandChanIfIndex
                      (5)odsiInBandChanPppIfIndex
                      (6)odsiInBandChanOdsiCpOperStatus
          (2)odsiInBandChanSonetObjects
                (1)odsiInBandChanSonetCount
                (2)odsiInBandChanSonetTable
                      (1)odsiInBandChanSonetEntry
                            (1)odsiInBandChanSonetSpec
          (3)odsiInBandChanDigitalWrapperObjects
                (1)odsiInBandChanDigitalWrapperCount
                (2)odsiInBandChanDigitalWrapperTable
                      (1)odsiInBandChanDigitalWrapperEntry
                            (1)odsiInBandChanDigitalWrapperSpec
    (4)odsiEndPointObjects
          (1)odsiEndPointCount
          (2)odsiEndPointTable
                (1)odsiEndPointEntry
                      (1)odsiEndPointUserGrp
                      (2)odsiEndPointIndex
                      (3)odsiEndPointPort
                      (4)odsiEndPointChanIfIndex
                      (5)odsiEndPointChanSpec
                      (6)odsiEndPointContainerIfIndex
                      (7)odsiEndPointContainerSpec
                      (8)odsiEndPointIpAddr
                      (9)odsiEndPointConnIndex
    (5)odsiConnObjects
          (1)odsiConnCount
          (2)odsiConnTable
                (1)odsiConnEntry
                      (1)odsiConnIndex
                      (2)odsiConnState
                      (3)odsiConnHeadEndPort
                      (4)odsiConnHeadEndChan
                      (5)odsiConnHeadEndContainer
                      (6)odsiConnHeadEndIpAddr
                      (7)odsiConnTailEndPort
                      (8)odsiConnTailEndChan
                      (9)odsiConnTailEndContainer
                      (10)odsiConnTailEndIpAddr
                      (11)odsiConnDeviceRole
                      (12)odsiConnLocEndPoint
                      (13)odsiConnUserGrp
                      (14)odsiConnContractId
                      (15)odsiConnHeadEndFraming
                      (16)odsiConnTailEndFraming
                      (17)odsiConnHeadEndTransparency
                      (18)odsiConnTailEndTransaprency
                      (19)odsiConnSizeRequested
                      (20)odsiConnSizeGranted
                      (21)odsiConnDirectionality
                      (22)odsiConnPropDelay
                      (23)odsiConnServiceLevel
                      (24)odsiConnDiversityCount
                      (25)odsiConnDirection
                      (26)odsiConnId
                      (27)odsiConnEstablishTime
                      (28)odsiConnDownReason
    (6)odsiSigObjects
          (1)odsiSigIpAddr
```

```
                (2)odsiSigConnectFailTimeout
                (3)odsiSigKeepAliveTimeout
                (4)odsiSigAfterLifeTimeout
                (5)odsiSigTxnDeadTimeout
        (4)odsiNotificationsPrefix
              (0) odsiNotifications
                  (1)odsiNewConnection
                  (2)odsiLostConnection
        (5)odsiConformance
              (1)odsiGroups
                    (1)odsiDeviceGroups
                          (1)odsiDeviceCommonGroup
                    (2)odsiPortGroups
                          (1)odsiPortCommonGroup
                          (2)odsiPortUserDeviceGroup
                    (3)odsiInBandChanGroups
                          (1)odsiInBandChanCommonGroup
                          (2)odsiInBandChanSonetGroup
                    (4)odsiEndPointGroups
                          (1)odsiEndPointUserDeviceGroup
                    (5)odsiConnGroups
                          (1)odsiConnCommonGroup
                    (6)odsiNotificationGroups
                          (1)odsiNotificationCommonGroup
               (2)odsiCompliances
                    (1)odsiCompliance
```

5.4 Groups

Objects in the ODSI MIB are arranged into groups. Each group consists of a set of closely related objects. The groups defined below map to the ODSI entities identified in Section 3.

- o odsiDeviceObjects
- o odsiPortObjects
- o odsiInBandChanObjects
- o odsiEndPointObjects
- o odsiConnObjects
- o odsiSigObjects
- o odsiNotifications

5.4.1 odsiDeviceObjects

The odsiDevice group consists of objects that pertain to the device as a whole. The objects in this group are:

o odsiDeviceProtocolVersion: The ODSI protocol version number implemented by this device.

o odsiDeviceType: The type of device, User Device or Network Device.

5.4.2 odsiPortObjects

The odsiPortObjects group consists of objects that pertain to ports on ODSI devices. Some of the prominent objects in this group are:

o odsiPortCount: The total number of ODSI ports in this device.

o odsiPortTable: A table used to manage ODSI ports.

o odsiPortStatsTable: A table that presents various statistics

associated with ODSI ports. The statistics include the number of connection requests received, number of connection requests made, etc.

o odsiPortLocAddrTable: A table used to assign IP addresses to local ODSI port.

o odsiPortRemAddrTable: A table used to associate IP addresses with ODSI ports on a remote (neighboring) device.

5.4.3 odsiInBandChanObjects

The odsiInBandChanObjects group consists of objects that pertain to in-band control channels on ODSI ports. Some of the prominent objects in this group are:

o odsiInBandChanTable: A table used to manage in-band control channels.

o odsiInBandChanSonetTable: A table used to manage the in-band control channel on SONET ports.

o odsiInBandChanDigitalWrapperTable: A table used to manage the in-band control channel on SONET ports.

5.4.4 odsiEndPointObjects

The odsiEndPoint group consists of objects that pertain to ODSI connection end points. Some of the prominent objects in this group are:

o odsiEndPointTable: A table implemented by user devices that is used to manage local ODSI end points.

5.4.5 odsiConnObjects

The odsiConnObjects group consists of objects that pertain to ODSI connections. Some of the prominent objects in this group are:

o odsiConnTable: A table implemented by user devices to monitor ODSI connections.

5.4.6 odsiSigObjects

The odsiSigObjects group consists of objects that pertain to ODSI signaling. The objects in this group are:

o odsiSigIpAddr: The IP address of this device used for initiaing and for receiving transport connections used for ODSI signaling.

o odsiSigConnectFailTimeout: The length of time that this device will wait for a transport connection to be established.

o odsiSigKeepAliveTimeout: The length of time between two successive KEEPALIVE messages sent by this device on a transport connection.

o odsiSigAfterLifeTimeout: The length of time that a transport connection is kept alive after all signaling transactions that reference the connection have terminated.

o odsiSigTxnDeadTimeout: The length of time that the state associated with a signaling transaction is retained while awaiting a response from the next control point.

o odsiConnTable: A table implemented by user devices to monitor ODSI connections.

5.4.7 odsiNotifications

- o odsiNewConnection: a notification issued when a connection is set up.
- o odsiLostConnection: a notification issued when a connection goes down.

5.5 Conformance Statements

The following groups are defined as units of conformance:

- o odsiDeviceCommonGroup: subset of odsiDeviceObjects that all ODSI devices are required to implement.
- o odsiPortCommonGroup: subset of odsiPortObjects that all ODSI devices are required to implement.
- o odsiPortUserDeviceGroup: subset of odsiPortObjects that ODSI user devices are required to implement.
- o odsiPortInBandChanGroup: subset of odsiInBandChanObjects that ODSI devices are required to implement, if they support in-band control channels.
- o odsiPortInBandChanSonetGroup: subset of odsiInBandChanObjects that ODSI devices are required to implement, if they support in-band control channels on SONET ports.
- o odsiEndPointUserDeviceGroup: subset of odsiEndPointObjects that ODSI user devices are required to implement.
- o odsiConnCommonGroup: list of odsiConnObjects that all ODSI devices are required to implement.
- o odsiSigCommonGroup: list of odsiSigObjects that all ODSI devices are required to implement.

The requirements stated for the above groups are captured in a compliance statement (odsiCompliance). This statment also relaxes the read-create requirement for objects in the odsiPortRemAddrTable to read-only, if a device does not support manual configuration of remote IP addresses.

5.6 Relationship to the MIB II Interfaces Group

The Interfaces group of MIB II [RFC2233] defines generic managed objects for managing interfaces. ODSI specifies a new type of interface, viz., ODSI Control Interface, used for in-band operation of ODSI protocols. This section specifies the extensions to the Interface group for managing ODSI Control Interfaces. Currently, extensions are defined only for the ODSI Control Interfaces defined over SONET and physical links employing digital wrapper framing.

5.6.1 Use of ifTable for the ODSI In-Band Control Channel

The In-Band Control Channel operates over a SONET (ifType=sonet(39)) interface or a physical link employing digital wrapper framing, and has a PPP interface layered over it.

o ifIndex: Interface index assigned to the in-band channel.

o ifDescr: ODSI In-Band Control Channel

o ifType: sonetOverheadChannel or digitalWrapperOverheadChannel.
These values are to be assigned by IANA. A value of
other(1) may be used until the assignments are received.

o ifSpeed: Speed of in-band channel (some multiple of 64 Kbps)

o ifPhysAddress: An OCTET STRING of zero length.

o ifAdminStatus: The desired administrative status of the
interface. Supports read-only access.

o ifOperStatus: as specified in [RFC2233].

o ifLastChange: as specified in [RFC2233].

o ifName: as specified in [RFC2233].

o ifLinkUpDownTrapEnable: Default value is enabled(1). Access may be
restricted to read-only.

o ifHighSpeed: Set to 0

o ifPromiscuousMode: Set to false(2)

o ifConnectorPresent: Set to false(2).

o ifAlias: as specified in [RFC2233]. Access may be restricted to
read-only.

6. Definitions

```
ODSI-MIB DEFINITIONS ::= BEGIN

IMPORTS
   MODULE-IDENTITY,
   OBJECT-IDENTITY,
   OBJECT-TYPE,
   NOTIFICATION-TYPE,
   Counter32,
   Gauge32,
   IpAddress,
   Unsigned32,
   enterprises
      FROM SNMPv2-SMI

   TEXTUAL-CONVENTION,
   RowStatus,
   TestAndIncr,
   TimeStamp,
   TruthValue
      FROM SNMPv2-TC

   MODULE-COMPLIANCE,
   NOTIFICATION-GROUP,
   OBJECT-GROUP
      FROM SNMPv2-CONF

   sysUpTime
```

```
      FROM SNMPv2-MIB

   InterfaceIndex,
   InterfaceIndexOrZero,
   ifIndex
      FROM IF-MIB;

odsiMibModule MODULE-IDENTITY
   LAST-UPDATED       "0007220000Z"
   ORGANIZATION       "Optical Domain Service Interconnect Coalition"
   CONTACT-INFO       "Editor: K. Arvind

                       Tenor Networks, Inc.
                       100 Nagog Park
                       Acton, MA 01720

                       Tel: +1 (978) 264-4900
                       Fax: +1 (978) 264-0671

                       email: arvind@tenornetworks.com"
   DESCRIPTION         "This module specifies a Management Information
                        Base for ODSI."

   REVISION           "0010090000Z"
   DESCRIPTION        "Rev 1.5 Oct 09, 2000
                       o Added odsiSigObjects and corresponding
                         compliance group and statement.
                       o Changed the description of odsiConnId to match
                         the 'Trail ID' attribute in the signaling spec.
                       o Added odsiConnId to the new and lost connection
                         notifications.
                       o Updated SYNTAX of odsiConnDeviceRole.
                       o Added compliance group and statement relating
                         to in-band control channels on ports that
                         employ digital wrappers.
                       o Changed SYNTAX of odsiPortLocalPortId from
                         Unsigned32 to InterfaceIndex.
                       o Changed SYNTAX of odsiPortRemotePortId from
                         Unsigned32 to InterfaceIndexOrZero.
                       o Changed SYNTAX of odsiPortLocAddrIndex from
                         Unsigned32 to PositiveInteger.
                       o Changed SYNTAX of odsiPortRemAddrIndex from
                         Unsigned32 to PositiveInteger.
                       o Changed SYNTAX of odsiConnDiversityIndex from
                         Unsigned32 to PositiveInteger.
                       o Changed enum values for odsiPortAuthAlgorithm
                         to start from 1 instead of 0.
                       o Changed enum values for odsiPortLocalMinChanSize
                         to start from 1 instead of 0.
                       o Changed enum values for odsiPortRemoteMinChanSize
                         to start from 1 instead of 0.
                       o Changed odsiCompliance to allow the following
                         parameters to be implemented as read-only:
                             - odsiPortLocalPortId
                             - odsiPortLocalMinChanSize
                             - odsiPortLocalConcatenation
                             - odsiPortLocAddrIpAddr
                             - odsiPortLocAddrStatus
                       o Changed odsiCompliance to remove the option of
                         implementing the following objects as read-only
                         (since a device that implements these as read-only
                          will not be able to interoperate with a device
                          that does not support in-band control channels):
                             - odsiPortRemotePortId
```

```
                        - odsiPortRemoteMinChanSize
                        - odsiPortRemoteConcatenation
                        - odsiPortRemAddrIpAddr
                        - odsiPortRemAddrUserGrp
                        - odsiPortRemAddrStatus
                      o Modified description of above objects to disallow
                        override via SNMP of values acquired via the
                        service discovery protocol, if an implementation
                        so chooses.

                    "

    REVISION        "0008120000Z"
    DESCRIPTION     "Rev 1.4 Aug 12, 2000
                      o Added objects odsiInBandChanPppIfIndex and
                        odsiInBandChanOdsiCpOperStatus.
                      o Changed the name of odsiInBandChanPresent to
                        odsiInBandChanSupported.
                      o Revised DESCRIPTION fields of objects
                        in the odsiInBandChanTable.
                      o Removed the odsiExperimental subtree, since
                        it didn't seem to be used at all.
                      o Added textual convention OdsiContainerSpec.
                      o Added odsiEndPointContainerIfIndex and
                        odsiEndPointContainerSpec.
                      o Removed the following columns from
                        odsiConnTable:
                        - odsiConnEncoding
                        - odsiConnPriority
                        - odsiConnRestoreTime
                        - odsiConnJitter
                        - odsiConnErrorRate
                        - odsiConnAvailability
                      o Added the following columns to odsiConnTable:
                        - odsiConnHeadEndContainer
                        - odsiConnTailEndContainer
                        - odsiConnId
                        - odsiConnContractId
                        - odsiConnDirectionality
                        - odsiConnServiceLevel
                        - odsiConnHeadEndFraming
                        - odsiConnTailEndFraming
                        - odsiConnHeadEndTransparency
                        - odsiConnTailEndTransparency
                        - odsiConnDiversityCount
                      o Added odsiConnDiversityTable
                      o Added textual convention OdsiConnFraming.
                      o Added textual convention OdsiConnTransparency.
                      o Removed textual convention OdsiPhysEncoding.
                      o Removed textual convention OdsiConnPriority.
                      o Removed subtree odsiPhysLayerEncodings.
                      o Added textual convention
                        OdsiInBandChanDigitalWrapperSpec.
                      o Added odsiInBandChanDigitalWrapperTable.
                      o Edited text preceding MIB module to reflect
                        above changes.
                    "

    REVISION        "0007220000Z"
    DESCRIPTION     "Rev 1.3 Jul 22, 2000
                      o Cleanup and changes mostly relating to support
                        for Address Registration and Service Discovery.

                      o Set root of odsi tree to IANA assigned value.
```

o Replaced the term subchannel with channel to be consistent with the functional spec.

o Changed the OdsiUserGroup textual convention to be based on an RFC 2685 VPN ID.

o odsiPortTable changes
  - Fixed a typo in the DESCRIPTION field of odsiPortEntry.

  - Removed odsiPortControlChan, and merged it with odsiInBandChanIfIndex.

  - Removed odsiPortUserGrpCount, since odsiPortUserGrpTable was removed.

  - Added the following new columns to the odsiPortTable:
    - odsiPortSigAddr
    - odsiPortAuthAlgorithm
    - odsiPortLocalPortId
    - odsiPortRemotePortId
    - odsiPortLocalMinChanSize
    - odsiPortRemoteMinChanSize
    - odsiPortLocalConcatenation
    - odsiPortRemoteConcatenation

o Changes to odsiPortLocAddrTable:
  - Updated DESCRIPTION field.
  - Added column odsiPortLocAddrUserGrp.

o Changes to odsiPortRemAddrTable:
  - Updated DESCRIPTION field.
  - Added column odsiPortRemAddrUserGrp.

o Removed odsiPortUserGrpNewIndex and odsiPortUserGrpTable, since user groups are associated with IP addresses and not ports, as per address registration and service discovery spec.

o Changes to odsiInBandChanTable:
  - Replaced INDEX clause of odsiInBandChanEntry with 'AUGMENTS { odsiPortEntry }'
  - Modified DESCRIPTION fields

o Added odsiNotificationsPrefix and moved odsiNotifications under this element to conform to SMIv2 style.

o odsiPortCommonGroup changes
  - Removed odsiPortUserGrpCount, since this object has been deleted from the MIB.
  - Added new object odsiPortLocAddrUserGrp.
  - Added new object odsiPortRemAddrUserGrp.
  - Removed odsiPortUserGrpNewIndex, odsiPortUserGrpValue and odsiPortUserGrpStatus, since the odsiUserGrpTable has been deleted from the MIB.

o Updated odsiCompliance to reflect above changes.

```
                        o Replaced lines of hyphens with lines of
                          equal signs, since MIB compilers can't handle
                          lines with an odd number of hyphens.
                       "

   REVISION            "0004250000Z"
   DESCRIPTION         "Rev 1.2 Apr 25, 2000
                        o Minor changes - text revised to use terminology
                          that is consistent with functional spec (terms
                          'user device' and 'network device' used in
                          place of 'client device' and 'optical network
                          device'.
                       "

   REVISION            "0004170000Z"
   DESCRIPTION         "Rev 1.1 Apr 17, 2000
                        o Incorporated feedback from the Chicago meeting.
                        o Terms 'Client Device' and 'Optical Network
                          Device' used in place of 'Electrical Device'
                          and 'Optical Device'.
                        o Added a new object called odsiConnDownReason to
                          convey the reason why a ODSI connection went
                          down. This object is included in the
                          odsiLostConnection notification.
                        o Enumeration value closed(6) deleted from the
                          OdsiConnState TEXTUAL-CONVENTION.
                        o Definitions of the textual convention
                          OdsiInBandChanSonetSpec and the MIB object
                          OdsiInBandChanSonetSpec updated to specify
                          the default value for the SONET In-Band
                          Channel.
                        o Syntax errors that prevented the MIB from
                          compiling fixed.
                       "

   REVISION            "0004050000Z"
   DESCRIPTION         "Rev 1.0 Apr 5, 2000
                        Initial version
                       "


   ::= { odsiModules 1 }


-- |=========================
-- | ODSI Textual Conventions
-- |=========================

PositiveInteger ::= TEXTUAL-CONVENTION
   STATUS              current
   DESCRIPTION
      "
      A 32-bit positive integer.
      "
   SYNTAX Unsigned32(1..4294967295)


OdsiUserGroup ::=  TEXTUAL-CONVENTION
   DISPLAY-HINT     "1x:"
   STATUS           current
   DESCRIPTION
      "
```

```
        An ODSI User Group maps directly to a Virtual Private Network
        (VPN) ID as defined in RFC 2685. The VPN ID consists of a 3
        octet VPN organizationally unique identifier (OUI) followed
        by a 4 octet VPN index. The VPN OUI identifies the VPN
        authority. The VPN index identifies a particular VPN serviced
        by the VPN authority.
       "
    SYNTAX OCTET STRING (SIZE (7))


OdsiDeviceType  ::= TEXTUAL-CONVENTION
    STATUS          current
    DESCRIPTION
        "
        A device may be a User Device (user(1)), or a Network Device
        (network(2)). ODSI connections are established between end points
        that reside on User Devices. User Devices connect to the
        optical network through Network Devices that are part of the
        edge of the optical network.
       "
    SYNTAX INTEGER {
              user(1),
              network(2)
           }


OdsiConnState  ::= TEXTUAL-CONVENTION
    STATUS          current
    DESCRIPTION
        "
        Various states an ODSI connection can be in:

        (NEEDS WORK)
        invalid(1): Unknown state
        other(2):   None of the following states
        calling(3): Connection is being set up
        open(4):    Connection set up is complete
        closing(5): Connection is being torn down
        "
    SYNTAX INTEGER {
             invalid(1),
             other(2),
             calling(3),
             open(4),
             closing(5)
           }


OdsiConnFraming ::= TEXTUAL-CONVENTION
    STATUS          current
    DESCRIPTION
        "
        Format of the signal on the port between user and network devices.
        Note that wanEthernet(6) is transported in an OC-192c.
        "
    SYNTAX INTEGER {
             pdh(1),
             sonet(2),
             sdh(3),
             digitalWrapper(4),
             lanEthernet(5),
```

```
            wanEthernet(6)
          }


OdsiConnTransparency ::= TEXTUAL-CONVENTION
   STATUS                current
   DESCRIPTION
      "
      The portion of the signal that is available for user data traffic.
      This parameter is interpreted with respect to the framing of the
      signal. For SONET/SDH framing, the transparency options are:

        o other(1): None of the following.
            This would be the value used if the framing is other than
            SONET/SDH.

        o plrC(2): Physical layer regeneration, meaning all the SONET
            signal is available for user data.

        o steC(3): End points are attached to section-terminating
            equipment. Only line and path overhead are available to
            user data traffic.

        o lteC(4): End points are attached to line-terminating
            equipment. Only path overhead is available to user data
            traffic.
      "
   SYNTAX INTEGER {
            other(1),
            plrC(2),
            steC(3),
            lteC(4)
          }


OdsiChannelSpec ::= TEXTUAL-CONVENTION
   STATUS              current
   DESCRIPTION
      "
      Identifies a channel on a port that supports channelization.
      A string of 0 length indicates that no channel is specified.
      The interpretation of this OCTET STRING depends on the ifType of
      the port with which the channel is associated:

         ifType=sonet(39):
            A channel is identified by the 'time slot' (position)
            that it occupies within the STS-N frame structure. The
            single level '1 to N in order of appearance at the input to
            the byte-interleaver' convention specified in Bellcore
            GR-253 section 6.1.2 (requirement R6-3) is used to specify
            the time slot.

            The number representing time slot is encoded in an ASCII
            string.  For example, STS-1 number 23 is encoded as the
            string '23'.

            For concatenated channels that are formed by concatenating
            several constituent channels, the encoding consists of
            a list of numbers instead of a single number. Each number
            in the list corresponds to the 'time slot' associated with
            its constituent channel. Each number is encoded in an
            ASCII string, and separated from other numbers in the list
```

```
            by a space.
            "
   SYNTAX OCTET STRING


OdsiContainerSpec ::= TEXTUAL-CONVENTION
   STATUS             current
   DESCRIPTION
      "
      Identifies a container on a channel (See OdsiChannelSpec) that
      supports division into containers. A string of 0 length indicates
      that no container is specified. The interpretation of this OCTET
      STRING depends on the ifType of the port with which the channel
      is associated:

         ifType=sonet(39):
           The OCTET STRING consists of a sequence of octet triples.
           Each octet triple identifies a virtual tributary that is
           part of the container. The first octet in a triple
           identifies the virtual tributary group number (1..7).
           The second octet identifies the size of VTs in the VT
           group identified by the first octet. The size can be one
           of the following:
             1: VT1.5/VC11
             2: VT2/VC12
             3: VT3
             4: VT6/VC3
           The third octet identifies the VT number within the VT group
           identified by the first octet. For example, the 4th VT1.5 in
           the 7th VT group is represented by the triple:
              (0x07, 0x01, 0x04).

           A container that is formed by the concatenation of the 3rd
           and 4th VT1.5 in the 7th VT group is represented by the
           following OCTET STRING:

              0x07, 0x01, 0x03, 0x07, 0x01, 0x04
      "
   SYNTAX OCTET STRING


OdsiInBandChanSonetSpec ::= TEXTUAL-CONVENTION
   STATUS                  current
   DESCRIPTION
      "
      Specifies the sequence of SONET overhead bytes that together
      constitute an in-band control channel.

      The sequence is specified as an OCTET STRING, in which each
      octet represents a specific SONET overhead byte. The mapping
      between octet values and the SONET overhead byte that they
      represent is given below:

           Value  Overhead Byte
           =====  ==============
           0x00     E1
           0x01     F1
           0x02     D1
           0x03     D2
           0x04     D3
           0x05     K1
           0x06     K2
```

```
                0x07      D4
                0x08      D5
                0x09      D6
                0x0a      D7
                0x0b      D8
                0x0c      D9
                0x0d      D10
                0x0e      D11
                0x0f      D12
                0x10      E2

            Example: A channel comprising of overhead bytes E1, F1, K2,
            and E2 would be represented by the octet string
                0x00, 0x01, 0x06, 0x10

            The recommended default sequence is D4-D12. This sequence is
            represented by the following octet string:
                0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f

          "
        SYNTAX OCTET STRING


    OdsiInBandChanDigitalWrapperSpec ::= TEXTUAL-CONVENTION
        STATUS                          current
        DESCRIPTION
            "
            Identifies one of the 4 10 Mbps overhead channels on a physical
            link employing digital wrappers, that is designated as the
            in-band control channel.

            The optical channel overhead (sometimes referred to as the user
            data channel) can be used to provide an in-band control channel
            for ODSI service discovery, address registration, and signaling.
            There are 40 Mbps of overhead bytes available for this purpose,
            which can be viewed as four 10 Mbps channels. Both user devices
            and network devices allow the configuration of one of these
            channels as an in-band control channel, which is encoded using
            10 Mbps full-duplex ethernet framing. This textual convention
            identifies the specific channel that is designated as the
            in-band control channel.

            A value of none indicates that an in-band control channel is not
            supported.
          "
        SYNTAX INTEGER {
                  none(0),
                  ch1(1),
                  ch2(2),
                  ch3(3),
                  ch4(4)
                }



    -- |==============
    -- | Registration
    -- |==============


    odsi            OBJECT-IDENTITY
        STATUS      current
```

```
   DESCRIPTION
      "
      The enterprises identifier assigned to ODSI by IANA. This defines
      the root of the ODSI object identifier space.
      "
   ::= { enterprises 6028 } --  Assigned by IANA to the ODSI Coalition


odsiModules    OBJECT-IDENTITY
   STATUS      current
   DESCRIPTION
      "
      All ODSI information modules are assigned an object identifier
      under this subtree.
      "
   ::= { odsi 1 }


odsiObjectIdentities OBJECT-IDENTITY
   STATUS            current
   DESCRIPTION
      "
      All ODSI object identities are defined under this subtree.
      "
   ::= { odsi 2 }


odsiObjects    OBJECT-IDENTITY
   STATUS      current
   DESCRIPTION
      "
      All ODSI objects are defined under this subtree.
      "
   ::= { odsi 3 }


odsiNotificationsPrefix OBJECT-IDENTITY
   STATUS               current
   DESCRIPTION
      "
      All ODSI notifications are defined under the
      odsiNotifications branch of this subtree.
      "
   ::= { odsi 4 }


odsiNotifications OBJECT-IDENTITY
   STATUS         current
   DESCRIPTION
      "
      All ODSI notifications are defined under this
      subtree.
      "
   ::= { odsiNotificationsPrefix 0 }


odsiConformance   OBJECT-IDENTITY
   STATUS         current
   DESCRIPTION
      "
      All information elements relating to ODSI MIB conformance are
      defined under this subtree.
      "
   ::= { odsi 5 }
```

```
-- |====================
-- | OBJECT IDENTITIES
-- |====================


-- |======================
-- | odsiProtocolVersions
-- |======================


odsiProtocolVersions OBJECT-IDENTITY
   STATUS            current
   DESCRIPTION
      "
      Object identifiers for ODSI protocol versions are defined under
      this subtree.
      "
   ::= { odsiObjectIdentities 1 }


odsiProtocolVersion0100 OBJECT-IDENTITY
   STATUS               current
   DESCRIPTION
      "
      Version 1.00 of the ODSI protocol suite.
      "
   ::= { odsiProtocolVersions 1 }




-- |============================================================
-- | odsiDevice Group
-- |   The odsiDevice group consists of objects that pertain
-- | to the device as a whole. This group consists of the
-- | following objects:
-- |
-- | o odsiDeviceProtocolVersion
-- | o odsiDeviceType
-- |============================================================

odsiDeviceObjects OBJECT IDENTIFIER ::= { odsiObjects 1 }

odsiDeviceProtocolVersion OBJECT-TYPE
   SYNTAX      OBJECT IDENTIFIER
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
      "
      The version of ODSI that this device supports. Object identifiers
      for ODSI versions are defined in the odsiProtocolVersion subtree.
      "

   ::= { odsiDeviceObjects 1 }


odsiDeviceType OBJECT-TYPE
   SYNTAX      OdsiDeviceType
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
```

```
      "
      Indicates if this device is a user device or a network device.
      "
   ::= { odsiDeviceObjects 2 }


-- |=========================================================
-- | odsiPort Group
-- |   The odsiPort group consists of objects that pertain
-- | to ports on ODSI devices.
-- |
-- | o odsiPortCount
-- | o odsiPortTable
-- | o odsiPortStatsTable
-- | o odsiPortLocAddrTable
-- | o odsiPortRemAddrTable
-- | o odsiPortUserGroupTable
-- |=========================================================

odsiPortObjects OBJECT IDENTIFIER ::= { odsiObjects 2 }

odsiPortCount  OBJECT-TYPE
   SYNTAX      Unsigned32
   MAX-ACCESS  read-only
   STATUS      current
   DESCRIPTION
      "
      The total number of ODSI-capable ports in the device.
      "

   ::= { odsiPortObjects 1 }


-- |================
-- | odsiPortTable
-- |================


odsiPortTable  OBJECT-TYPE
   SYNTAX      SEQUENCE OF OdsiPortEntry
   MAX-ACCESS  not-accessible
   STATUS      current
   DESCRIPTION
      "
      A table for managing ODSI-capable ports in the device. The number
      of entries in the table is specified by the object odsiPortCount.
      "

   ::= { odsiPortObjects 2 }


odsiPortEntry  OBJECT-TYPE
   SYNTAX      OdsiPortEntry
   MAX-ACCESS  not-accessible
   STATUS      current
   DESCRIPTION
      "
      An entry in the odsiPortTable. The entry is identified by
      the ifIndex associated with the physical interface corresponding
      to the entry. A row in this table cannot be created or deleted by
      SNMP operations on columns of the table.
      "
   INDEX           { ifIndex }
```

```
   ::= { odsiPortTable 1 }

OdsiPortEntry ::= SEQUENCE {
   odsiPortAdminStatus            INTEGER,
   odsiPortOperStatus             INTEGER,
   odsiPortNumChannels            Unsigned32,
   odsiPortLocalAddrCount         Unsigned32,
   odsiPortRemoteAddrCount        Unsigned32,
   odsiPortSigAddr                IpAddress,
   odsiPortAuthAlgorithm          INTEGER,
   odsiPortLocalPortId            InterfaceIndex,
   odsiPortRemotePortId           InterfaceIndexOrZero,
   odsiPortLocalMinChanSize       INTEGER,
   odsiPortRemoteMinChanSize      INTEGER,
   odsiPortLocalConcatenation     INTEGER,
   odsiPortRemoteConcatenation    INTEGER
}

odsiPortAdminStatus OBJECT-TYPE
   SYNTAX            INTEGER {
                        up(1),
                        down(2)
                     }
   MAX-ACCESS        read-write
   STATUS            current
   DESCRIPTION
      "
      The desired administrative status for ODSI on this port. A value
      of up(1) indicates that ODSI is to be enabled on this port. A
      value of down(2) indicates that ODSI is to be disabled on this
      port.
      "

   ::=  { odsiPortEntry 1 }


odsiPortOperStatus  OBJECT-TYPE
   SYNTAX            INTEGER {
                        up(1),
                        down(2)
                     }
   MAX-ACCESS        read-only
   STATUS            current
   DESCRIPTION
      "
      The operational status of ODSI on this port. A value of up(1)
      indicates that ODSI is enabled on this port. A value of down(2)
      indicates that ODSI is disabled on this port.
      "

   ::=  { odsiPortEntry 2 }


odsiPortNumChannels    OBJECT-TYPE
   SYNTAX              Unsigned32
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
     "
     Indicates the maximum number of channels supported by this
```

```
      port. A value of 0 indicates that this port does not support
      channelization. The in-band control channel (if present) is
      not included in this count.
      "

   ::= { odsiPortEntry 3 }


odsiPortLocalAddrCount OBJECT-TYPE
   SYNTAX              Unsigned32
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
      "
      The number of entries in the odsiPortLocAddrTable that are
      associated with this port. The value of this object is typically
      1 for UNI type applications.
      "

   ::= { odsiPortEntry 4 }


odsiPortRemoteAddrCount OBJECT-TYPE
   SYNTAX              Unsigned32
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
      "
      The number of entries in the odsiPortRemAddrTable that are
      associated with this port. The value of this object is typically
      1 for UNI type applications.
      "

   ::= { odsiPortEntry 5 }


odsiPortSigAddr OBJECT-TYPE
   SYNTAX       IpAddress
   MAX-ACCESS   read-write
   STATUS       current
   DESCRIPTION
      "
      The IP address used by a user device as the destination of
      signaling requests. If the service discovery protocol
      is supported on this port, a network device places the value
      of this object in the 'Signaling-Address' option of ODSICP
      Configuration Request packets, while a user device receives the
      value of this object from the 'Signaling-Address' option.
      "

   ::= { odsiPortEntry 6 }


odsiPortAuthAlgorithm OBJECT-TYPE
   SYNTAX              INTEGER {
                         none(1),
                         md5(2)
                       }
   MAX-ACCESS          read-write
   STATUS              current
   DESCRIPTION
      "
```

```
      The authentication algorithm used by the network device to
      authenticate addresses registered on this port. If the service
      discovery protocol is supported on this port, a network device
      places the value of this object in the 'Authentication-algorithm'
      option of ODSICP Configuration Request packets, while a user device
      receives the value of this object from the same option.
      "

   ::= { odsiPortEntry 7 }


odsiPortLocalPortId  OBJECT-TYPE
   SYNTAX            InterfaceIndex
   MAX-ACCESS        read-write
   STATUS            current
   DESCRIPTION
      "
      A value that uniquely identifies this port within the device.
      The value may be identical to the ifIndex associated with
      this port. If the service discovery protocol is supported on
      this port, the value of this object is placed in the
      'Port-ID' option of ODSICP Configuration Request packets that are
      transmitted on this port.
      "

   ::= { odsiPortEntry 8 }



odsiPortRemotePortId  OBJECT-TYPE
   SYNTAX             InterfaceIndexOrZero
   MAX-ACCESS         read-write
   STATUS             current
   DESCRIPTION
      "
      A value that uniquely identifies the port within the remote
      device to which this port is attached. If the service discovery
      protocol is supported on this port, the value of this object is
      received in the 'Port-ID' option of ODSICP Configuration Request
      packets that are received on this port. An implementation may
      choose to not allow a value acquired through the service discovery
      protocol to be overridden via SNMP. A value of 0 indicates
      that the remote port id is not known.
      "

   ::= { odsiPortEntry 9 }


odsiPortLocalMinChanSize OBJECT-TYPE
   SYNTAX                INTEGER {
                           notChannelized(1),
                           sts1(2),
                           oc3(3),
                           oc12(4),
                           oc48(5),
                           oc192(6),
                           oc768(7)
                         }
   MAX-ACCESS            read-write
   STATUS                current
   DESCRIPTION
      "
      If channelization is supported on this port, this object
      specifies the minimum size of each channel. If the service
```

```
      discovery protocol is supported on this port, the value of
      this object is placed in the 'Sub-channel-size' option of
      ODSICP Configuration Request packets that are transmitted
      on this port.
      "

   ::= { odsiPortEntry 10 }


odsiPortRemoteMinChanSize OBJECT-TYPE
   SYNTAX                 INTEGER {
                            notChannelized(1),
                            sts1(2),
                            oc3(3),
                            oc12(4),
                            oc48(5),
                            oc192(6),
                            oc768(7)
                          }
   MAX-ACCESS             read-write
   STATUS                 current
   DESCRIPTION
      "
      If channelization is supported on the remote device port
      to which this port is attached, this object specifies the
      minimum size of each channel. If the service discovery
      protocol is supported on this port, the value of
      this object is received in the 'Sub-channel-size' option of
      ODSICP Configuration Request packets that are received
      on this port. An implementation may choose to not allow a
      value acquired through the service discovery protocol to be
      overridden via SNMP.
      "

   ::= { odsiPortEntry 11 }



odsiPortLocalConcatenation OBJECT-TYPE
   SYNTAX                 INTEGER {
                            none(0),
                            standard(1),
                            arbitrary(2)
                          }
   MAX-ACCESS             read-write
   STATUS                 current
   DESCRIPTION
      "
      If channelization is supported on this port, this object
      indicates the ability of the port to concatenate channels
      together into a single logical channel. If the service
      discovery protocol is supported on this port, the value of
      this object is placed in the 'Concatenation' option of
      ODSICP Configuration Request packets that are transmitted
      on this port.
      "

   ::= { odsiPortEntry 12 }


odsiPortRemoteConcatenation OBJECT-TYPE
   SYNTAX                 INTEGER {
```

```
                                none(0),
                                standard(1),
                                arbitrary(2)
                              }
    MAX-ACCESS                read-write
    STATUS                    current
    DESCRIPTION
        "
        If channelization is supported on this port, this object
        indicates the ability of the remote device port to concatenate
        channels together into a single logical channel. If the service
        discovery protocol is supported on this port, the value of
        this object is received in the 'Concatenation' option of
        ODSICP Configuration Request packets that are received
        on this port. An implementation may choose to not allow a value
        acquired through the service discovery protocol to be overridden
        via SNMP.
        "

    ::= { odsiPortEntry 13 }



-- |====================
-- | odsiPortStatsTable
-- |====================


odsiPortStatsTable  OBJECT-TYPE
    SYNTAX            SEQUENCE OF OdsiPortStatsEntry
    MAX-ACCESS        not-accessible
    STATUS            current
    DESCRIPTION
        "
        A table of per-port ODSI protocol statistics. The maximum number
        of entries in this table is given by odsiPortCount.
        "

    ::= { odsiPortObjects 3 }


odsiPortStatsEntry  OBJECT-TYPE
    SYNTAX            OdsiPortStatsEntry
    MAX-ACCESS        not-accessible
    STATUS            current
    DESCRIPTION
        "
        An entry in the odsiPortStatsTable. The entry is identified by
        ifIndex associated with the port corresponding to the entry.
        A row in this table cannot be created or deleted by SNMP
        operations on columns of the table.
        "
    INDEX           { ifIndex }

    ::= { odsiPortStatsTable 1 }


OdsiPortStatsEntry ::= SEQUENCE
{
    odsiPortStatsInCalls          Counter32,
    odsiPortStatsInCallRefusals   Counter32,
    odsiPortStatsInCallResets     Counter32,
    odsiPortStatsInConnections    Counter32,
```

```
   odsiPortStatsOutCallAttempts  Counter32,
   odsiPortStatsOutCallFailures  Counter32,
   odsiPortStatsOutCallResets    Counter32,
   odsiPortStatsOutConnections   Counter32
}

odsiPortStatsInCalls OBJECT-TYPE
   SYNTAX            Counter32
   MAX-ACCESS        read-only
   STATUS            current
   DESCRIPTION
      "
      The number of incoming connection requests received for this port.
      "

   ::=  { odsiPortStatsEntry 1 }

odsiPortStatsInCallRefusals OBJECT-TYPE
   SYNTAX                   Counter32
   MAX-ACCESS               read-only
   STATUS                   current
   DESCRIPTION
      "
      The number of incoming connection requests received for this port
      that were refused.
      "

   ::=  { odsiPortStatsEntry 2 }

odsiPortStatsInCallResets OBJECT-TYPE
   SYNTAX                  Counter32
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      The number of incoming connection requests received for this port
      that were reset.
      "

   ::=  { odsiPortStatsEntry 3 }

odsiPortStatsInConnections OBJECT-TYPE
   SYNTAX                   Counter32
   MAX-ACCESS               read-only
   STATUS                   current
   DESCRIPTION
      "
      The number of active incoming connections received for this port.
      This includes connections whose setup is still in progress.
      "

   ::=  { odsiPortStatsEntry 4 }

odsiPortStatsOutCallAttempts OBJECT-TYPE
   SYNTAX                    Counter32
   MAX-ACCESS                read-only
   STATUS                    current
   DESCRIPTION
      "
```

```
      The number of outgoing connection requests initiated from this
      port.
      "

   ::=  { odsiPortStatsEntry 5 }


odsiPortStatsOutCallFailures OBJECT-TYPE
   SYNTAX                    Counter32
   MAX-ACCESS                read-only
   STATUS                    current
   DESCRIPTION
      "
      The number of outgoing connection requests initiated from this
      port that failed to complete.
      "

   ::=  { odsiPortStatsEntry 6 }


odsiPortStatsOutCallResets OBJECT-TYPE
   SYNTAX                   Counter32
   MAX-ACCESS               read-only
   STATUS                   current
   DESCRIPTION
      "
      The number of outgoing connection requests initiated from this port
      that were reset.
      "

   ::=  { odsiPortStatsEntry 7 }


odsiPortStatsOutConnections OBJECT-TYPE
   SYNTAX                   Counter32
   MAX-ACCESS               read-only
   STATUS                   current
   DESCRIPTION
      "
      The number of active outgoing connections initiated from this
      port. This includes connections whose setup is still in progress.
      "

   ::=  { odsiPortStatsEntry 8 }



-- |======================
-- |  odsiPortLocAddrTable
-- |======================


odsiPortLocAddrNewIndex OBJECT-TYPE
   SYNTAX               TestAndIncr
   MAX-ACCESS           read-write
   STATUS               current
   DESCRIPTION
       "
       This object is used to assign values to the object
       odsiPortLocAddrIndex as described in 'Textual Conventions
```

```
        for SMIv2'. The network manager reads the object, and then
        writes the value back in the SET that creates a new instance of
        odsiPortLocAddrEntry. If the SET fails with the code
        'inconsistentValue', then the process must be repeated. If the
        SET succeeds, then the object is incremented, and the new
        instance is created according to the manager's specifications.
        "

    ::= { odsiPortObjects 4 }


odsiPortLocAddrTable OBJECT-TYPE
    SYNTAX              SEQUENCE OF OdsiPortLocAddrEntry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "
        A table that lists the IP addresses assigned to ODSI ports, and
        the user groups that these addresses are associated with. The
        information in this table is used by the ODSI address
        registration protocol to register addresses with the remote
        devices to which this device is attached.

        The number of entries in the table associated with any port
        is specified by the object odsiPortLocAddrCount. There is
        typically 1 entry for each ODSI-capable port in the device.
        However, multiple entries may be present in NNI-type
        applications of ODSI.
        "

    ::= { odsiPortObjects 5 }


odsiPortLocAddrEntry OBJECT-TYPE
    SYNTAX              OdsiPortLocAddrEntry
    MAX-ACCESS          not-accessible
    STATUS              current
    DESCRIPTION
        "
        An entry in the odsiPortLocAddrTable. The table employs two
        indices:
        (1) ifIndex: The ifIndex associated with the ODSI port.
        (2) odsiPortLocAddrIndex: a unique value associated with each
            entry in the table.

        A row in this table can be created or deleted by SNMP operations
        on columns of the table.
        "

    INDEX { ifIndex, odsiPortLocAddrIndex }

    ::= { odsiPortLocAddrTable 1 }


OdsiPortLocAddrEntry ::= SEQUENCE {
    odsiPortLocAddrIndex   PositiveInteger,
    odsiPortLocAddrIpAddr  IpAddress,
    odsiPortLocAddrUserGrp OdsiUserGroup,
    odsiPortLocAddrStatus  RowStatus
}


odsiPortLocAddrIndex OBJECT-TYPE
```

```
      SYNTAX            PositiveInteger
      MAX-ACCESS        not-accessible
      STATUS            current
      DESCRIPTION
         "
         A number that unambiguously identifies this entry.
         "

      ::= { odsiPortLocAddrEntry 1 }


   odsiPortLocAddrIpAddr OBJECT-TYPE
      SYNTAX            IpAddress
      MAX-ACCESS        read-create
      STATUS            current
      DESCRIPTION
          "
          The IP address associated with this entry. If the port associated
          with this entry supports the address registration protocol, this
          address is is placed in the 'Registered-Addresses' option of
          ODSICP Register-Address packets that are transmitted on this
          port.
          "

      ::= { odsiPortLocAddrEntry 2 }


   odsiPortLocAddrUserGrp OBJECT-TYPE
      SYNTAX            OdsiUserGroup
      MAX-ACCESS        read-create
      STATUS            current
      DESCRIPTION
          "
          The user group to which the IP address associated with this
          entry belongs. If the port associated with this entry supports
          the address registration protocol, this address is is placed
          in the 'User-Group' option of ODSICP Register-Address packets
          that are transmitted on this port.
          "

      ::= { odsiPortLocAddrEntry 3 }


   odsiPortLocAddrStatus OBJECT-TYPE
      SYNTAX            RowStatus
      MAX-ACCESS        read-create
      STATUS            current
      DESCRIPTION
          "
          This object is used to create a new row or delete an existing row
          in this table.
          "

      ::= { odsiPortLocAddrEntry 4 }




   -- |======================
   -- | odsiPortRemAddrTable
   -- |======================
```

```
odsiPortRemAddrNewIndex OBJECT-TYPE
   SYNTAX               TestAndIncr
   MAX-ACCESS           read-write
   STATUS               current
   DESCRIPTION
        "
        This object is used to assign values to the object
        odsiPortRemAddrIndex as described in 'Textual Conventions
        for SMIv2'. The network manager reads the object, and then
        writes the value back in the SET that creates a new instance of
        odsiPortRemAddrEntry. If the SET fails with the code
        'inconsistentValue', then the process must be repeated. If the
        SET succeeds, then the object is incremented, and the new
        instance is created according to the manager's specifications.
        "

  ::= { odsiPortObjects 6 }


odsiPortRemAddrTable OBJECT-TYPE
   SYNTAX            SEQUENCE OF OdsiPortRemAddrEntry
   MAX-ACCESS        not-accessible
   STATUS            current
   DESCRIPTION
        "
        A table that lists the IP addresses assigned to ports on the
        remote device to which ODSI ports on this device are attached,
        and the user groups that these addresses are associated with.

        The number of entries in the table associated with any port
        is specified by the object odsiPortRemAddrCount. There is
        typically 1 entry for each ODSI-capable port in the device.
        However, multiple entries may be present in NNI-type
        applications of ODSI.
        "

    ::= { odsiPortObjects 7 }


odsiPortRemAddrEntry OBJECT-TYPE
   SYNTAX            OdsiPortRemAddrEntry
   MAX-ACCESS        not-accessible
   STATUS            current
   DESCRIPTION
        "
        An entry in the odsiPortRemAddrTable. The table employs two
        indices:
        (1) ifIndex: The ifIndex associated with the ODSI port on the
            local device.
        (2) odsiPortRemAddrIndex: a unique value associated with each
            entry in the table.

        A row is typically created when IP address information is
        received from the remote device as a result of running the ODSI
        Address Registration protocol on the in-band control channel
        associated with this port.

        A row in this table can also be created or deleted by SNMP
        operations on columns of the table. This is typically done when
        the port does not support an in-band control channel.
        "
```

```
   INDEX { ifIndex, odsiPortRemAddrIndex }

   ::= { odsiPortRemAddrTable 1 }


OdsiPortRemAddrEntry ::= SEQUENCE {
   odsiPortRemAddrIndex   PositiveInteger,
   odsiPortRemAddrIpAddr  IpAddress,
   odsiPortRemAddrUserGrp OdsiUserGroup,
   odsiPortRemAddrStatus  RowStatus
}


odsiPortRemAddrIndex OBJECT-TYPE
   SYNTAX             PositiveInteger
   MAX-ACCESS         not-accessible
   STATUS             current
   DESCRIPTION
      "
      A number that unambiguosly identifies this entry.
      "

   ::= { odsiPortRemAddrEntry 1 }


odsiPortRemAddrIpAddr OBJECT-TYPE
   SYNTAX             IpAddress
   MAX-ACCESS         read-create
   STATUS             current
   DESCRIPTION
      "
      The IP address associated with this entry. If the port associated
      with this entry supports the address registration protocol, this
      address is received in the 'Registered-Addresses' option of
      ODSICP Register-Address packets that are received on this
      port. An implementation may choose to not allow a value acquired
      through the address registration protocol to be overridden via
      SNMP.
      "

   ::= { odsiPortRemAddrEntry 2 }


odsiPortRemAddrUserGrp OBJECT-TYPE
   SYNTAX             OdsiUserGroup
   MAX-ACCESS         read-create
   STATUS             current
   DESCRIPTION
      "
      The user group to which the IP address associated with this
      entry belongs. If the port associated with this entry supports
      the address registration protocol, this value is received in
      the 'User-Group' option of ODSICP Register-Address packets that
      are received on this port. An implementation may choose to not
      allow a value acquired through the service discovery protocol to
      be overridden via SNMP.
      "

   ::= { odsiPortRemAddrEntry 3 }
```

```
odsiPortRemAddrStatus OBJECT-TYPE
   SYNTAX              RowStatus
   MAX-ACCESS          read-create
   STATUS              current
   DESCRIPTION
      "
      This object is used to create a new row or delete an existing row
      in this table. An implementation may choose to not allow a value
      acquired through the service discovery protocol to be deleted via
      SNMP.
      "

   ::= { odsiPortRemAddrEntry 4 }




-- |=======================
-- | odsiInBandChanObjects
-- |=======================

odsiInBandChanObjects OBJECT-IDENTITY
   STATUS              current
   DESCRIPTION
      "
      Objects relating to ODSI in-band control channels are assigned
      under this subtree.
      "
   ::= { odsiObjects 3 }



-- |=====================
-- | odsiInBandChanTable
-- |=====================


odsiInBandChanTable OBJECT-TYPE
   SYNTAX            SEQUENCE OF OdsiInBandChanEntry
   MAX-ACCESS        not-accessible
   STATUS            current
   DESCRIPTION
      "
      A table for managing in-band control channels associated with
      ODSI-capable ports in the device. The number of entries in the
      table is specified by the object odsiPortCount.
      "

   ::= { odsiInBandChanObjects 1 }


odsiInBandChanEntry OBJECT-TYPE
   SYNTAX            OdsiInBandChanEntry
   MAX-ACCESS        not-accessible
   STATUS            current
   DESCRIPTION
      "
      An entry in the odsiInBandChanTable. The entry is identified by
      ifIndex associated with the physical interface corresponding to
      the entry. A row in this table cannot be created or deleted by
```

```
      SNMP operations on columns of the table.
      "
   AUGMENTS        { odsiPortEntry }

   ::= { odsiInBandChanTable 1 }


OdsiInBandChanEntry ::= SEQUENCE {
   odsiInBandChanSupported        TruthValue,
   odsiInBandChanAdminStatus      INTEGER,
   odsiInBandChanOperStatus       INTEGER,
   odsiInBandChanIfIndex          InterfaceIndexOrZero,
   odsiInBandChanPppIfIndex       InterfaceIndexOrZero,
   odsiInBandChanOdsiCpOperStatus INTEGER
}



odsiInBandChanSupported OBJECT-TYPE
   SYNTAX               TruthValue
   MAX-ACCESS           read-only
   STATUS               current
   DESCRIPTION
      "
      Indicates if an ODSI in-band control channel is supported on
      the physical interface corresponding to this ODSI port. If an
      in-band channel is supported, the entry in a physical layer
      specific table corresponding to this port may be used to manage
      the channel. Physical layer-specific tables are defined for the
      following ifType values:

          sonet(39) - odsiInBandChanSonetTable

      "

   ::= { odsiInBandChanEntry 1 }



odsiInBandChanAdminStatus OBJECT-TYPE
   SYNTAX                INTEGER {
                            up(1),
                            down(2)
                         }
   MAX-ACCESS            read-write
   STATUS                current
   DESCRIPTION
      "
      The desired administrative status for the in-band control channel
      on this port.

      A value of up(1) indicates that the in-band channel is enabled on
      this port. When the in-band channel is enabled, the interface
      corresponding to it is brought up.

      A value of down(2) indicates that the in-band channel is
      disabled. When the in-band channel is disabled, the interface
      corresponding to it is brought down.

      For ports that do not support an in-band control channel,
      setting the value of this object has no effect.
```

```
      "
   ::= { odsiInBandChanEntry 2 }


odsiInBandChanOperStatus OBJECT-TYPE
   SYNTAX                  INTEGER {
                              up(1),
                              down(2)
                           }
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      The current operational status of the in-band control channel on
      this port.

      A value of up(1) indicates that the in-band channel is
      operational. The object odsiInBandChanIfIndex contains the
      ifIndex of the interface associated with the in-band channel in
      this state.

      A value of down(2) indicates that the in-band channel is not
      operational. The object odsiInBandChanIfIndex assumes the value
      0 in this case.

      For ports that do not support an in-band control interface,
      this object always has the value down(2).
      "
   ::= { odsiInBandChanEntry 3 }


odsiInBandChanIfIndex OBJECT-TYPE
   SYNTAX                  InterfaceIndexOrZero
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      The ifIndex of the in-band control channel associated with this
      port.  If the port does not support an in-band control channel,
      this object has a value of zero.
      "

   ::= { odsiInBandChanEntry 4 }


odsiInBandChanPppIfIndex OBJECT-TYPE
   SYNTAX                  InterfaceIndexOrZero
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      The ifIndex of the PPP interface than is stacked over the in-band
      control channel associated with this port.  If the port does not
      support an in-band control channel, or if a PPP interface over the
      in-band control channel does not exist, this object has a value
      of zero.
      "

   ::= { odsiInBandChanEntry 5 }
```

```
odsiInBandChanOdsiCpOperStatus OBJECT-TYPE
   SYNTAX                      INTEGER {
                                  up(1),
                                  down(2)
                               }
   MAX-ACCESS                  read-only
   STATUS                      current
   DESCRIPTION
      "
      The current operational status of the ODSI service discovery
      (PPP ODSI NCP) protocol on the in-band control channel on
      this port.

      A value of up(1) indicates that the PPP ODSI NCP is in the
      OPENED state.

      A value of down(2) indicates that the PPP ODSI NCP is not in
      the OPENED state.

      For ports that do not support an in-band control interface, or
      if a PPP interface has not been established over the in-band
      control interface, this object always has the value down(2).
      "
   ::= { odsiInBandChanEntry 6 }




-- |============================
-- | odsiInBandChanSonetObjects
-- |============================

odsiInBandChanSonetObjects OBJECT-IDENTITY
   STATUS                  current
   DESCRIPTION
      "
      Objects relating to ODSI in-band control channels on SONET ports
      are assigned under this subtree.
      "
   ::= { odsiInBandChanObjects 2 }


-- |==========================
-- | odsiInBandChanSonetTable
-- |==========================

odsiInBandChanSonetCount OBJECT-TYPE
   SYNTAX                  Unsigned32
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      Number of ODSI SONET in-band control channels on this device.
      "
   ::= { odsiInBandChanSonetObjects 1 }


odsiInBandChanSonetTable OBJECT-TYPE
   SYNTAX                  SEQUENCE OF OdsiInBandChanSonetEntry
```

```
   MAX-ACCESS              not-accessible
   STATUS                  current
   DESCRIPTION
      "
      A table that lists all the ODSI SONET in-band control channels
      on this device.

      The number of entries in this table is determined by the total
      number of ODSI-capable SONET ports supported by the device, and is
      given by  odsiInBandChanSonetCount.

      "
   ::= { odsiInBandChanSonetObjects 2 }


odsiInBandChanSonetEntry OBJECT-TYPE
   SYNTAX                  OdsiInBandChanSonetEntry
   MAX-ACCESS              not-accessible
   STATUS                  current
   DESCRIPTION
      "
      An entry in the odsiInBandChanSonetTable. Entries are indexed by
      the ifIndex of the ODSI port on which this channel is multiplexed.

      Entries in this table cannot be created or deleted using SNMP Set
      operations on column objects in this table.
      "
   INDEX { ifIndex }

   ::= { odsiInBandChanSonetTable 1 }



OdsiInBandChanSonetEntry ::= SEQUENCE {
   odsiInBandChanSonetSpec OdsiInBandChanSonetSpec
}


odsiInBandChanSonetSpec OBJECT-TYPE
   SYNTAX                  OdsiInBandChanSonetSpec
   MAX-ACCESS              read-write
   STATUS                  current
   DESCRIPTION
      "
      The sequence of SONET overhead bytes that constitute the in-band
      control channel defined by this entry.

      Unless explicitly overridden, the channel is constituted from the
      following overhead bytes: D4-D12. This corresponds to the
      following OCTET STRING value for this object:
      '0708090a0b0c0d0e0f'h.
      "

   ::=  { odsiInBandChanSonetEntry 1 }



-- |===================================
-- |  odsiInBandChanDigitalWrapperObjects
-- |===================================
```

```
odsiInBandChanDigitalWrapperObjects OBJECT-IDENTITY
   STATUS                          current
   DESCRIPTION
      "
      Objects relating to ODSI in-band control channels on ports
      that employ digital wrappers are assigned under this subtree.
      "
   ::= { odsiInBandChanObjects 3 }


-- |=================================
-- | odsiInBandChanDigitalWrapperTable
-- |=================================


odsiInBandChanDigitalWrapperCount OBJECT-TYPE
   SYNTAX                          Unsigned32
   MAX-ACCESS                      read-only
   STATUS                          current
   DESCRIPTION
      "
      Number of ODSI in-band control channels on this device on links
      that employ digital wrappers.
      "
   ::= { odsiInBandChanDigitalWrapperObjects 1 }


odsiInBandChanDigitalWrapperTable OBJECT-TYPE
   SYNTAX            SEQUENCE OF OdsiInBandChanDigitalWrapperEntry
   MAX-ACCESS        not-accessible
   STATUS            current
   DESCRIPTION
      "
      A table that lists all the ODSI in-band control channels
      on this device on interfaces that employ digital wrappers.

      The number of entries in this table is determined by the total
      number of ODSI-capable ports supported by the device that employ
      digital wrappers, and is given by  the MIB object
      odsiInBandChanDigitalWrapperCount.
      "
   ::= { odsiInBandChanDigitalWrapperObjects 2 }


odsiInBandChanDigitalWrapperEntry OBJECT-TYPE
   SYNTAX                          OdsiInBandChanDigitalWrapperEntry
   MAX-ACCESS                      not-accessible
   STATUS                          current
   DESCRIPTION
      "
      An entry in the odsiInBandChanDigitalWrapperTable. Entries are
      indexed by the ifIndex of the ODSI port on which this control
      channel is multiplexed.

      Entries in this table cannot be created or deleted using SNMP Set
      operations on column objects in this table.
      "
   INDEX { ifIndex }

   ::= { odsiInBandChanDigitalWrapperTable 1 }
```

```
OdsiInBandChanDigitalWrapperEntry ::= SEQUENCE {
   odsiInBandChanDigitalWrapperSpec OdsiInBandChanDigitalWrapperSpec
}


odsiInBandChanDigitalWrapperSpec OBJECT-TYPE
   SYNTAX                        OdsiInBandChanDigitalWrapperSpec
   MAX-ACCESS                    read-write
   STATUS                        current
   DESCRIPTION
      "
      The 10 Mbps overhead channel that is designated as the in-band
      control channel defined by this entry.
      "

   ::=  { odsiInBandChanDigitalWrapperEntry 1 }




-- |====================
-- | odsiEndPointObjects
-- |====================


odsiEndPointObjects OBJECT-IDENTITY
   STATUS           current
   DESCRIPTION
      "
      Objects relating to ODSI connection end points are assigned under
      this subtree.
      "
   ::= { odsiObjects 4 }



odsiEndPointCount OBJECT-TYPE
   SYNTAX         Unsigned32
   MAX-ACCESS     read-only
   STATUS         current
   DESCRIPTION
      "
      The total number of ODSI connection end points on this device.
      This object is implemented by user devices only.
      "
   ::= { odsiEndPointObjects 1 }



odsiEndPointTable OBJECT-TYPE
   SYNTAX         SEQUENCE OF OdsiEndPointEntry
   MAX-ACCESS     not-accessible
   STATUS         current
   DESCRIPTION
      "
      A table that lists all the ODSI connection end points on this
      device associated with various ODSI user groups.

      The number of entries in this table is determined by the total
      number of channels supported by the device, and is given by
      odsiEndPointCount.
```

```
      This object is implemented by user devices only.
      "
   ::= { odsiEndPointObjects 2 }



odsiEndPointEntry OBJECT-TYPE
   SYNTAX         OdsiEndPointEntry
   MAX-ACCESS     not-accessible
   STATUS         current
   DESCRIPTION
      "
      An entry in the odsiEndPointTable. Entries are indexed by two
      indices:

      (1) odsiEndPointUserGrp: The ODSI User group associated with the
          entry.
      (2) odsiEndPointIndex: An identifier that uniquely identifies this
          entry.

      Entries in this table cannot be created or deleted using SNMP Set
      operations on column objects in this table.
      "
   INDEX { odsiEndPointUserGrp, odsiEndPointIndex }
   ::= { odsiEndPointTable 1 }



OdsiEndPointEntry ::= SEQUENCE
{
   odsiEndPointUserGrp          OdsiUserGroup,
   odsiEndPointIndex            PositiveInteger,
   odsiEndPointPort             InterfaceIndex,
   odsiEndPointChanIfIndex      InterfaceIndexOrZero,
   odsiEndPointChanSpec         OdsiChannelSpec,
   odsiEndPointContainerIfIndex InterfaceIndexOrZero,
   odsiEndPointContainerSpec    OdsiContainerSpec,
   odsiEndPointIpAddr           IpAddress,
   odsiEndPointConnIndex        Unsigned32
}



odsiEndPointUserGrp OBJECT-TYPE
   SYNTAX           OdsiUserGroup
   MAX-ACCESS       not-accessible
   STATUS           current
   DESCRIPTION
      "
      The ODSI User Group associated with this entry.
      "

   ::= { odsiEndPointEntry 1 }



odsiEndPointIndex OBJECT-TYPE
   SYNTAX         PositiveInteger
   MAX-ACCESS     not-accessible
   STATUS         current
   DESCRIPTION
      "
```

```
      An index that uniquely identifies this entry.
      "

   ::= { odsiEndPointEntry 2 }


odsiEndPointPort OBJECT-TYPE
   SYNTAX        InterfaceIndex
   MAX-ACCESS    read-only
   STATUS        current
   DESCRIPTION
      "
      The ifIndex of the port that is associated with this end point.
      "

   ::= { odsiEndPointEntry 3 }


odsiEndPointChanIfIndex OBJECT-TYPE
   SYNTAX               InterfaceIndexOrZero
   MAX-ACCESS           read-only
   STATUS               current
   DESCRIPTION
      "
      The ifIndex of the interface corresponding to the channel
      associated with this end point.

      For an end point associated with a port that is not channelized,
      this object assumes the value 0.
      "

   ::= { odsiEndPointEntry 4 }


odsiEndPointChanSpec OBJECT-TYPE
   SYNTAX            OdsiChannelSpec
   MAX-ACCESS        read-only
   STATUS            current
   DESCRIPTION
      "
      Specifies the channel with which this end point is associated
      in a physical-layer specific manner.
      "

   ::= { odsiEndPointEntry 5 }


odsiEndPointContainerIfIndex OBJECT-TYPE
   SYNTAX                    InterfaceIndexOrZero
   MAX-ACCESS                read-only
   STATUS                    current
   DESCRIPTION
      "
      The ifIndex of the interface corresponding to the container
      associated with this end point.

      For an end point associated with a channel that does not
      support division into containers, this object assumes the value
      0.
      "
```

```
   ::= { odsiEndPointEntry 6 }


odsiEndPointContainerSpec OBJECT-TYPE
   SYNTAX                 OdsiContainerSpec
   MAX-ACCESS             read-only
   STATUS                 current
   DESCRIPTION
      "
      Specifies the container with which this end point is associated
      in a physical-layer specific manner.
      "

   ::= { odsiEndPointEntry 7 }


odsiEndPointIpAddr OBJECT-TYPE
   SYNTAX          IpAddress
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      The IP address associated with this end point. This value may not
      be meaningful, if the end point is not associated with an active
      connection, i.e., if odsiEndPointConnIndex is 0.
      "

   ::= { odsiEndPointEntry 8 }


odsiEndPointConnIndex OBJECT-TYPE
   SYNTAX             Unsigned32
   MAX-ACCESS         read-only
   STATUS             current
   DESCRIPTION
      "
      Pointer to entry in the odsiConnTable that describes the
      connection with which this end point is associated. If the end
      point is not associated with any connection (i.e., it is
      available), this object assumes the value 0.
      "

   ::= { odsiEndPointEntry 9 }



-- |=================
-- | odsiConnObjects
-- |=================


odsiConnObjects OBJECT-IDENTITY
   STATUS       current
   DESCRIPTION
      "
      Objects relating to ODSI connections (bandwidth channels) are
      assigned under this subtree.
      "
   ::= { odsiObjects 5 }
```

```
odsiConnCount   OBJECT-TYPE
   SYNTAX       Unsigned32
   MAX-ACCESS   read-only
   STATUS       current
   DESCRIPTION
      "
      The total number of ODSI connections (bandwidth channels) on this
      device.
      "
   ::= { odsiConnObjects 1 }


odsiConnTable OBJECT-TYPE
   SYNTAX     SEQUENCE OF OdsiConnEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
      "
      A table that lists all the ODSI connections (bandwidth channels)
      on this device.

      The maximum number of entries in this table is determined by the
      total number of channels supported by the device. The current
      number of entries is given by odsiConnNumConnections.
      "
   ::= { odsiConnObjects 2 }



odsiConnEntry OBJECT-TYPE
   SYNTAX     OdsiConnEntry
   MAX-ACCESS not-accessible
   STATUS     current
   DESCRIPTION
      "
      An entry in the odsiConnTable. Entries are indexed by
      odsiConnIndex, a unique identifier associated with each entry.

      Entries in this table cannot be created and deleted using SNMP Set
      operations on column objects in this table.
      "
   INDEX { odsiConnIndex }

   ::= { odsiConnTable 1 }


OdsiConnEntry ::= SEQUENCE
{
   odsiConnIndex               PositiveInteger,
   odsiConnState               OdsiConnState,
   odsiConnHeadEndPort         InterfaceIndexOrZero,
   odsiConnHeadEndChan         OdsiChannelSpec,
   odsiConnHeadEndContainer    OdsiContainerSpec,
   odsiConnHeadEndIpAddr       IpAddress,
   odsiConnTailEndPort         InterfaceIndexOrZero,
   odsiConnTailEndChan         OdsiChannelSpec,
   odsiConnTailEndContainer    OdsiContainerSpec,
   odsiConnTailEndIpAddr       IpAddress,
   odsiConnDeviceRole          INTEGER,
   odsiConnLocEndPoint         Unsigned32,
   odsiConnUserGrp             OdsiUserGroup,
   odsiConnContractId          OCTET STRING,
   odsiConnHeadEndFraming      OdsiConnFraming,
   odsiConnTailEndFraming      OdsiConnFraming,
```

```
   odsiConnHeadEndTransparency OdsiConnTransparency,
   odsiConnTailEndTransparency OdsiConnTransparency,
   odsiConnSizeRequested       Gauge32,
   odsiConnSizeGranted         Gauge32,
   odsiConnDirectionality      INTEGER,
   odsiConnPropDelay           Unsigned32,
   odsiConnServiceLevel        Unsigned32,
   odsiConnDiversityCount      Unsigned32,
   odsiConnDirection           INTEGER,
   odsiConnId                  Unsigned32,
   odsiConnEstablishTime       TimeStamp,
   odsiConnDownReason          INTEGER
}


odsiConnIndex  OBJECT-TYPE
   SYNTAX      PositiveInteger
   MAX-ACCESS  not-accessible
   STATUS      current
   DESCRIPTION
      "
      Unique index that identifies this entry.
      "

   ::= { odsiConnEntry 1 }



odsiConnState  OBJECT-TYPE
   SYNTAX      OdsiConnState
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
      "
      Current state of the connection.
      "
  ::= { odsiConnEntry 2 }



odsiConnHeadEndPort OBJECT-TYPE
   SYNTAX           InterfaceIndexOrZero
   MAX-ACCESS       read-only
   STATUS           current
   DESCRIPTION
      "
      The ifIndex of the port at the head end of the connection. The
      object assumes a value of 0, if the ifIndex is unknown.

      If this device is not physically part of the connection (e.g.,
      a third party device that establishes a connection between two
      other devices), one of the end points is arbitrarily designated
      as the head end and the other as the tail end; otherwise head
      end refers to the device that initiated the connection, and tail
      end refers to the device associated with the other end point.
      "

   ::= { odsiConnEntry 3 }



odsiConnHeadEndChan OBJECT-TYPE
   SYNTAX           OdsiChannelSpec
   MAX-ACCESS       read-only
```

```
   STATUS            current
   DESCRIPTION
      "
      The channel associated with the end point at the head end of
      the connection. An empty value indicates that the channel is
      not known or defined.

      If this device is not physically part of the connection (e.g.,
      a third party device that establishes a connection between two
      other devices), one of the end points is arbitrarily designated
      as the head end and the other as the tail end. otherwise head
      end refers to the device that initiated the connection, and tail
      end refers to the device associated with the other end point.
      "
   ::= { odsiConnEntry 4 }


odsiConnHeadEndContainer OBJECT-TYPE
   SYNTAX                OdsiContainerSpec
   MAX-ACCESS            read-only
   STATUS                current
   DESCRIPTION
      "
      The container associated with the end point at the head end of
      the connection. An empty value indicates that the container is
      not known or defined.

      If this device is not physically part of the connection (e.g.,
      a third party device that establishes a connection between two
      other devices), one of the end points is arbitrarily designated
      as the head end and the other as the tail end. otherwise head
      end refers to the device that initiated the connection, and tail
      end refers to the device associated with the other end point.
      "
   ::= { odsiConnEntry 5 }


odsiConnHeadEndIpAddr OBJECT-TYPE
   SYNTAX             IpAddress
   MAX-ACCESS         read-only
   STATUS             current
   DESCRIPTION
      "
      The IP address associated with the end point at the head end of
      the connection.

      If this device is not physically part of the connection (e.g.,
      a third party device that establishes a connection between two
      other devices), one of the end points is arbitrarily designated
      as the head end and the other as the tail end. otherwise head
      end refers to the device that initiated the connection, and tail
      end refers to the device associated with the other end point.
      "
   ::= { odsiConnEntry 6 }


odsiConnTailEndPort OBJECT-TYPE
   SYNTAX           InterfaceIndexOrZero
   MAX-ACCESS       read-only
   STATUS           current
   DESCRIPTION
```

```
        "
        The ifIndex of the port at the tail end of the connection. The
        object assumes a value of 0, if the ifIndex is unknown.

        If this device is not physically part of the connection (e.g.,
        a third party device that establishes a connection between two
        other devices), one of the end points is arbitrarily designated
        as the head end and the other as the tail end. otherwise head
        end refers to the device that initiated the connection, and tail
        end refers to the device associated with the other end point.
        "

    ::= { odsiConnEntry 7 }


odsiConnTailEndChan OBJECT-TYPE
    SYNTAX          OdsiChannelSpec
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
        "
        The channel associated with the end point at the tail end of
        the connection. An empty value indicates that the channel is
        not known or defined.

        If this device is not physically part of the connection (e.g.,
        a third party device that establishes a connection between two
        other devices), one of the end points is arbitrarily designated
        as the head end and the other as the tail end. otherwise head
        end refers to the device that initiated the connection, and tail
        end refers to the device associated with the other end point.
        "
    ::= { odsiConnEntry 8 }


odsiConnTailEndContainer OBJECT-TYPE
    SYNTAX              OdsiContainerSpec
    MAX-ACCESS          read-only
    STATUS              current
    DESCRIPTION
        "
        The container associated with the end point at the tail end of
        the connection. An empty value indicates that the container is
        not known or defined.

        If this device is not physically part of the connection (e.g.,
        a third party device that establishes a connection between two
        other devices), one of the end points is arbitrarily designated
        as the head end and the other as the tail end. otherwise head
        end refers to the device that initiated the connection, and tail
        end refers to the device associated with the other end point.
        "
    ::= { odsiConnEntry 9 }


odsiConnTailEndIpAddr OBJECT-TYPE
    SYNTAX          IpAddress
    MAX-ACCESS      read-only
    STATUS          current
    DESCRIPTION
```

```
      "
      The IP address associated with the end point at the tail end of
      the connection.

      If this device is not physically part of the connection (e.g.,
      a third party device that establishes a connection between two
      other devices), one of the end points is arbitrarily designated
      as the head end and the other as the tail end. otherwise head
      end refers to the device that initiated the connection, and tail
      end refers to the device associated with the other end point.
      "
   ::= { odsiConnEntry 10 }


odsiConnDeviceRole OBJECT-TYPE
   SYNTAX          INTEGER {
                      requester(1),
                      headEnd(2),
                      requesterAndHeadEnd(3),
                      tailEnd(4),
                      onc(5),
                      network(6)
                   }
   MAX-ACCESS   read-only
   STATUS       current
   DESCRIPTION
      "
      The role of the device with respect to this connection:

      requester(1): This device initiated the set up of this connection
      as a third party requestor.

      headEnd(2): This device constitutes the head end of the
      connection.

      requesterAndHeadEnd(3): This device initiated the set up of this
      connection and is also the head end of the connection.

      tailEnd(4): This device constitutes the tail end of the
      connection.

      onc(5): This device is or forms part of the optical network
      controller.

      network(6): This device is an optical network edge device to which
      the user device at the head end or tail end of this connection is
      attached.
      "
   ::= { odsiConnEntry 11 }


odsiConnLocEndPoint OBJECT-TYPE
   SYNTAX          Unsigned32
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      The odsiEndPointIndex corresponding to the end point associated
      with this connection, if an end point of the connection resides
      on this device. If this device does not contain an end point of
      this device, this object assumes the value 0.
      "
```

```
   ::= { odsiConnEntry 12 }


odsiConnUserGrp OBJECT-TYPE
   SYNTAX       OdsiUserGroup
   MAX-ACCESS   read-only
   STATUS       current
   DESCRIPTION
      "
      The ODSI user group associated with the connection.
      "
   ::= { odsiConnEntry 13 }


odsiConnContractId OBJECT-TYPE
   SYNTAX          OCTET STRING
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      A carrier-assigned identification that identifies the service
      contract. This parameter is included in bandwidth requests, and
      provides billing and authorization information for the connection.
      "
   ::= { odsiConnEntry 14 }


odsiConnHeadEndFraming OBJECT-TYPE
   SYNTAX              OdsiConnFraming
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
      "
      Format of the signal on the port between user and network
      devices at the head end of the connection.
      "
   ::= { odsiConnEntry 15 }


odsiConnTailEndFraming OBJECT-TYPE
   SYNTAX              OdsiConnFraming
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
      "
      Format of the signal on the port between user and network
      devices at the tail end of the connection.
      "
   ::= { odsiConnEntry 16 }


odsiConnHeadEndTransparency OBJECT-TYPE
   SYNTAX                  OdsiConnTransparency
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      Indicates the portion of the signal that is available for user
      data traffic on the port between user and network devices at the
      head end.
```

```
      "
   ::= { odsiConnEntry 17 }


odsiConnTailEndTransparency OBJECT-TYPE
   SYNTAX                   OdsiConnTransparency
   MAX-ACCESS               read-only
   STATUS                   current
   DESCRIPTION
      "
      Indicates the portion of the signal that is available for user
      data traffic on the port between user and network devices at the
      tail end.
      "
   ::= { odsiConnEntry 18 }


odsiConnSizeRequested OBJECT-TYPE
   SYNTAX             Gauge32
   MAX-ACCESS         read-only
   STATUS             current
   DESCRIPTION
      "
      The connection bandwidth requested by the initiator of the
      connection in millions of bits per second. The actual bandwidth
      granted may be larger than the requested bandwidth, if the
      requested size is not supported.
      "
   ::= { odsiConnEntry 19 }


odsiConnSizeGranted OBJECT-TYPE
   SYNTAX           Gauge32
   MAX-ACCESS       read-only
   STATUS           current
   DESCRIPTION
      "
      The bandwidth of the connection in millions of bits per
      second. This may be larger than the bandwidth requested by the
      initiator of the connection, if the requested size is not
      supported.
      "
   ::= { odsiConnEntry 20 }


odsiConnDirectionality OBJECT-TYPE
   SYNTAX               INTEGER {
                          uniDirectional(1),
                          biDirectional(2)
                        }
   MAX-ACCESS           read-only
   STATUS               current
   DESCRIPTION
      "
      Indicates if this connection is uni-directional or bi-directional.
      "
   ::= { odsiConnEntry 21 }
```

```
odsiConnPropDelay OBJECT-TYPE
   SYNTAX          Unsigned32
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      The maximum amount of propagation delay in milliseconds to be
      tolerated on the connection. A value of 0 is interpreted to mean
      that the propagation delay is unspecified.
      "
   ::= { odsiConnEntry 22 }


odsiConnServiceLevel OBJECT-TYPE
   SYNTAX             Unsigned32
   MAX-ACCESS         read-only
   STATUS             current
   DESCRIPTION
      "
      An integer specifying the service level requested for the
      connection. Service level encompasses priority (whether the
      connection can be preempted by other, higher priority
      connections), protection (whether the bandwidth should be
      protected against failures, and if so, the speed at which the
      protection will restore bandwidth service after a failure), and
      availability. Service level is encoded as an integer, whose
      meaning is assigned by the optical network provider. For example,
      one provider could encode its 'gold service' as service level 1,
      which might be advertised by the provider as '50 millisec restoral
      through BLSR technology'.
      "
   ::= { odsiConnEntry 23 }


odsiConnDiversityCount OBJECT-TYPE
   SYNTAX               Unsigned32
   MAX-ACCESS           read-only
   STATUS               current
   DESCRIPTION
      "
      The number of ODSI-created connections from which physical
      diversity was requested for this connection. This number also
      represents the number of entries in the odsiConnDiversityTable
      that correspond to this connection.
      "
   ::= { odsiConnEntry 24 }


odsiConnDirection OBJECT-TYPE
   SYNTAX          INTEGER {
                      outgoing(1),
                      incoming(2),
                      both(3),
                      neither(4)
                   }
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      The direction of the connection with respect to this device.
```

```
      outgoing(1): The connection was initiated by this user
      device, and an end point of the connection resides on this device.

      incoming(2): The connection was not initiated by this user
      device, but an end point of the connection resides on this device.

      both(3): The connection includes this device on its path, but
      neither end point of the connection resides on this device.

      neither(4): The connection was initiated by this third party
      device, but does not include this device on its path.
      "
   ::= { odsiConnEntry 25 }


odsiConnId      OBJECT-TYPE
   SYNTAX       Unsigned32
   MAX-ACCESS   read-only
   STATUS       current
   DESCRIPTION
      "
      A unique identifier assigned by the head end device to this
      bandwidth connection. This identifier is referred to as the
      'Trail ID' in the ODSI signaling specification.
      "
   ::= { odsiConnEntry 26 }


odsiConnEstablishTime OBJECT-TYPE
   SYNTAX             TimeStamp
   MAX-ACCESS         read-only
   STATUS             current
   DESCRIPTION
      "
       The time at which setup for this connection complete.
      "
   ::= { odsiConnEntry 27 }


odsiConnDownReason OBJECT-TYPE
   SYNTAX          INTEGER {
                     none(1),
                     other(2)
                     -- more codes to be added
                   }
   MAX-ACCESS      read-only
   STATUS          current
   DESCRIPTION
      "
      The reason why this connection went down. This object always
      returns a value of none(1), which indicates that the connection
      is still up. The only reason for the existence of this object
      is to convey the reason why a connection was reset in a SNMP
      notification (see odsiLostConnection). The reasons why a
      connection may be reset are:

      other(2): a reason other than one of those listed below.

      "
   ::= { odsiConnEntry 28 }
```

```
-- |=======================
-- | odsiConnDiversityTable
-- |=======================


odsiConnDiversityTable OBJECT-TYPE
   SYNTAX              SEQUENCE OF OdsiConnDiversityEntry
   MAX-ACCESS          not-accessible
   STATUS              current
   DESCRIPTION
       "
       A table that lists the ODSI connections from which a given
       ODSI connection is to be diverse.

       The number of entries in the table associated with any connection
       is specified by the object odsiConnDiversityCount.
       "

   ::= { odsiConnObjects 3 }



odsiConnDiversityEntry OBJECT-TYPE
   SYNTAX              OdsiConnDiversityEntry
   MAX-ACCESS          not-accessible
   STATUS              current
   DESCRIPTION
       "
       An entry in the odsiConnDiversityTable. The table employs
       two indices:
       (1) odsiConnIndex: The ODSI connection with which this entry is
           associated.
       (2) odsiConnDiversityIndex: a unique value associated with
           each entry in the table.

       A row in this table cannot be created or deleted by SNMP
       operations on columns of the table.
       "

   INDEX { odsiConnIndex, odsiConnDiversityIndex }

   ::= { odsiConnDiversityTable 1 }


OdsiConnDiversityEntry ::= SEQUENCE {
   odsiConnDiversityIndex       PositiveInteger,
   odsiConnDiversityConnDescr   OCTET STRING,
   odsiConnDiversityType        INTEGER
}


odsiConnDiversityIndex OBJECT-TYPE
   SYNTAX              PositiveInteger
   MAX-ACCESS          not-accessible
   STATUS              current
   DESCRIPTION
       "
       A number that unambiguosly identifies this entry.
       "
```

```
   ::= { odsiConnDiversityEntry 1 }

odsiConnDiversityConnDescr OBJECT-TYPE
   SYNTAX                  OCTET STRING(SIZE(8))
   MAX-ACCESS              read-only
   STATUS                  current
   DESCRIPTION
      "
      A unique identifier within the optical network that identifies
      the ODSI-created connection that the connection associated with
      this entry must be diverse from.

      The identifier is encoded in 8 octets. The first 4 octets consist
      of the 4 bytes (order: most significant to least significant) of
      the IP address associated with one of the connection end points.
      The last 4 octets consist of the 4 bytes (order: most significant
      to least significant) of a 32-bit ID assigned by the Optical
      Network to the connection.
      "

   ::= { odsiConnDiversityEntry 2 }


odsiConnDiversityType OBJECT-TYPE
   SYNTAX              INTEGER {
                          linkDiverse(1),
                          nodeDiverse(2),
                          srlgDiverse(3)
                       }
   MAX-ACCESS          read-only
   STATUS              current
   DESCRIPTION
      "
      The type of diversity specified.
      "

   ::= { odsiConnDiversityEntry 3 }


-- |===============
-- | odsiSigObjects
-- |===============


odsiSigObjects  OBJECT-IDENTITY
   STATUS       current
   DESCRIPTION
      "
      Objects relating to ODSI signaling are
      assigned under this subtree.
      "
   ::= { odsiObjects 6 }


odsiSigIpAddr   OBJECT-TYPE
   SYNTAX       IpAddress
   MAX-ACCESS   read-only
   STATUS       current
   DESCRIPTION
      "
```

```
      The IP Address of the device. This is used for initiating and
      receiving TCP connections. It is common practice for devices
      which use TCP as a transport to use a loop-back IP interface to
      receive TCP connection requests as this makes the device more
      resilient to the failure of any single interface.
      "
   ::= { odsiSigObjects 1 }


odsiSigConnectFailTimeout OBJECT-TYPE
   SYNTAX                 Unsigned32
   MAX-ACCESS             read-write
   STATUS                 current
   DESCRIPTION
      "
      The length of time that this device will wait for a TCP
      connection to be established to a peer. The value chosen
      should be sufficiently large to allow TCP initialization.
      "
   ::= { odsiSigObjects 2 }


odsiSigKeepAliveTimeout OBJECT-TYPE
   SYNTAX                 Unsigned32
   MAX-ACCESS             read-write
   STATUS                 current
   DESCRIPTION
      "
      The length of time between two successive KEEPALIVE messages
      sent by this device on a transport connection. The default
      value for this time interval is 30 seconds.
      "
   ::= { odsiSigObjects 3 }


odsiSigAfterLifeTimeout OBJECT-TYPE
   SYNTAX                 Unsigned32
   MAX-ACCESS             read-write
   STATUS                 current
   DESCRIPTION
      "
      The length of time for which a transport connection is retained
      after all signaling transactions that reference the connection
      have terminated.
      "
   ::= { odsiSigObjects 4 }


odsiSigTxnDeadTimeout OBJECT-TYPE
   SYNTAX              Unsigned32
   MAX-ACCESS          read-write
   STATUS              current
   DESCRIPTION
      "
      The length of time for which the state associated with a
      signaling transaction is retained while awaiting a response from
      the next control point.
      "
   ::= { odsiSigObjects 5 }
```

```
-- |===================
-- | odsiNotifications
-- |===================

odsiNewConnection NOTIFICATION-TYPE
   OBJECTS          {
                       odsiConnHeadEndPort,
                       odsiConnHeadEndChan,
                       odsiConnHeadEndIpAddr,
                       odsiConnTailEndPort,
                       odsiConnTailEndChan,
                       odsiConnTailEndIpAddr,
                       odsiConnUserGrp,
                       odsiConnId,
                       sysUpTime
                    }
   STATUS           current
   DESCRIPTION
      "
      A new ODSI connection that involves this device in some role
      has been set up.
      "
   ::= { odsiNotifications 0 1 }


odsiLostConnection NOTIFICATION-TYPE
   OBJECTS          {
                       odsiConnHeadEndPort,
                       odsiConnHeadEndChan,
                       odsiConnHeadEndIpAddr,
                       odsiConnTailEndPort,
                       odsiConnTailEndChan,
                       odsiConnTailEndIpAddr,
                       odsiConnUserGrp,
                       odsiConnDownReason,
                       sysUpTime
                    }
   STATUS           current
   DESCRIPTION
      "
      An existing ODSI connection has gone down.
      "
   ::= { odsiNotifications 0 2 }



-- |===================
-- | odsiConformance
-- |===================



-- |============
```

```
-- | odsiGroups
-- |============


odsiGroups OBJECT-IDENTITY
  STATUS   current
  DESCRIPTION
      "
      Groups that are defined as units of conformance are assiged uner
      this subtree.
      "
  ::= { odsiConformance 1 }


odsiDeviceGroups OBJECT-IDENTITY
  STATUS   current
  DESCRIPTION
      "
      Groups consisting of ODSI device-related objects are assigned under
      this subtree.
      "
  ::= { odsiGroups 1 }


odsiDeviceCommonGroup OBJECT-GROUP
   OBJECTS            {
                         odsiDeviceProtocolVersion,
                         odsiDeviceType
                      }
   STATUS             current
   DESCRIPTION
      "
      The objects for device-wide management.
      "
   ::= { odsiDeviceGroups 1 }


odsiPortGroups OBJECT-IDENTITY
  STATUS   current
  DESCRIPTION
      "
      Groups consisting of ODSI port-related objects are assigned under
      this subtree.
      "
  ::= { odsiGroups 2 }


odsiPortCommonGroup OBJECT-GROUP
   OBJECTS          {
                       odsiPortCount,
                       odsiPortAdminStatus,
                       odsiPortOperStatus,
                       odsiPortNumChannels,
                       odsiPortLocalAddrCount,
                       odsiPortRemoteAddrCount,
                       odsiPortLocAddrNewIndex,
                       odsiPortLocAddrIpAddr,
                       odsiPortLocAddrUserGrp,
                       odsiPortLocAddrStatus,
                       odsiPortRemAddrNewIndex,
                       odsiPortRemAddrIpAddr,
                       odsiPortRemAddrUserGrp,
```

```
                    odsiPortRemAddrStatus
                  }
    STATUS        current
    DESCRIPTION
       "
       The objects for ODSI port management that are implemented by
       both user and network devices.
       "
    ::= { odsiPortGroups 1 }


odsiPortUserDeviceGroup OBJECT-GROUP
    OBJECTS              {
                            odsiPortStatsInCalls,
                            odsiPortStatsInCallRefusals,
                            odsiPortStatsInCallResets,
                            odsiPortStatsOutCallAttempts,
                            odsiPortStatsOutCallFailures,
                            odsiPortStatsOutCallResets,
                            odsiPortStatsOutConnections,
                            odsiPortStatsInConnections
                         }
    STATUS               current
    DESCRIPTION
       "
       The objects for ODSI port management that are implemented by
       user devices only.
       "
    ::= { odsiPortGroups 2 }


odsiInBandChanGroups OBJECT-IDENTITY
   STATUS   current
   DESCRIPTION
      "
      Groups consisting of ODSI in-band control channel-related objects
      are assigned under this subtree.
      "
   ::= { odsiGroups 3 }


odsiInBandChanCommonGroup OBJECT-GROUP
    OBJECTS              {
                            odsiInBandChanSupported,
                            odsiInBandChanAdminStatus,
                            odsiInBandChanOperStatus,
                            odsiInBandChanIfIndex,
                            odsiInBandChanPppIfIndex,
                            odsiInBandChanOdsiCpOperStatus
                         }
    STATUS               current
    DESCRIPTION
       "
       The objects for ODSI in-band channel management that are
       implemented by both user and network devices.
       "
    ::= { odsiInBandChanGroups 1 }


odsiInBandChanSonetGroup OBJECT-GROUP
```

```
   OBJECTS                 {
                              odsiInBandChanSonetCount,
                              odsiInBandChanSonetSpec
                           }
   STATUS                  current
   DESCRIPTION
      "
      The objects for ODSI in-band control channel management that are
      implemented by SONET devices that support in-band control
      channels.
      "
   ::= { odsiInBandChanGroups 2 }


odsiInBandChanDigitalWrapperGroup OBJECT-GROUP
   OBJECTS                 {
                              odsiInBandChanDigitalWrapperCount,
                              odsiInBandChanDigitalWrapperSpec
                           }
   STATUS                  current
   DESCRIPTION
      "
      The objects for ODSI in-band control channel management that are
      implemented by devices that support in-band control channels on
      physical links that employ digital wrappers.
      "
   ::= { odsiInBandChanGroups 3 }


odsiEndPointGroups OBJECT-IDENTITY
  STATUS           current
  DESCRIPTION
     "
     Groups consisting of ODSI end point-related objects
     are assigned under this subtree.
     "
  ::= { odsiGroups 4 }


odsiEndPointUserDeviceGroup OBJECT-GROUP
   OBJECTS                  {
                               odsiEndPointCount,
                               odsiEndPointPort,
                               odsiEndPointChanIfIndex,
                               odsiEndPointChanSpec,
                               odsiEndPointIpAddr,
                               odsiEndPointConnIndex
                            }
   STATUS                   current
   DESCRIPTION
      "
      The objects for ODSI end point management that are
      implemented by user devices only.
      "
   ::= { odsiEndPointGroups 1 }


odsiConnGroups OBJECT-IDENTITY
  STATUS       current
```

```
  DESCRIPTION
      "
      Groups consisting of ODSI connection-related objects
      are assigned under this subtree.
      "
  ::= { odsiGroups 5 }


odsiConnCommonGroup OBJECT-GROUP
   OBJECTS          {
                       odsiConnState,
                       odsiConnHeadEndPort,
                       odsiConnHeadEndChan,
                       odsiConnHeadEndIpAddr,
                       odsiConnTailEndPort,
                       odsiConnTailEndChan,
                       odsiConnTailEndIpAddr,
                       odsiConnDeviceRole,
                       odsiConnLocEndPoint,
                       odsiConnUserGrp,
                       odsiConnContractId,
                       odsiConnHeadEndFraming,
                       odsiConnTailEndFraming,
                       odsiConnHeadEndTransparency,
                       odsiConnTailEndTransparency,
                       odsiConnSizeRequested,
                       odsiConnSizeGranted,
                       odsiConnDirectionality,
                       odsiConnPropDelay,
                       odsiConnServiceLevel,
                       odsiConnDiversityCount,
                       odsiConnDirection,
                       odsiConnId,
                       odsiConnEstablishTime,
                       odsiConnDownReason,
                       odsiConnDiversityConnDescr,
                       odsiConnDiversityType
                    }
    STATUS          current
    DESCRIPTION
       "
       The objects for ODSI connection management that are
       implemented by both user and network devices.
       "
    ::= { odsiConnGroups 1 }


odsiSigGroups OBJECT-IDENTITY
  STATUS       current
  DESCRIPTION
      "
      Groups consisting of ODSI signaling-related objects
      are assigned under this subtree.
      "
  ::= { odsiGroups 6 }


odsiSigCommonGroup OBJECT-GROUP
   OBJECTS          {
```

```
                    odsiSigIpAddr,
                    odsiSigConnectFailTimeout,
                    odsiSigKeepAliveTimeout,
                    odsiSigAfterLifeTimeout,
                    odsiSigTxnDeadTimeout
                }
    STATUS          current
    DESCRIPTION
        "
        The objects for ODSI connection management that are
        implemented by both user and network devices.
        "
    ::= { odsiSigGroups 1 }



odsiNotificationGroups OBJECT-IDENTITY
    STATUS    current
    DESCRIPTION
        "
        Groups consisting of ODSI notification-related objects
        are assigned under this subtree.
        "
    ::= { odsiGroups 7 }



odsiNotificationCommonGroup NOTIFICATION-GROUP
    NOTIFICATIONS           {
                                odsiNewConnection,
                                odsiLostConnection
                            }
    STATUS                  current
    DESCRIPTION
        "
        Notifications that are implemented by both user and
        network devices.
        "
    ::= { odsiNotificationGroups 1 }



-- |==================
-- |  odsiCompliances
-- |==================


odsiCompliances OBJECT-IDENTITY
    STATUS        current
    DESCRIPTION
        "
        ODSI compliance statements are defined under this subtree.
        "
    ::= { odsiConformance 2 }



-- |=================
-- |  odsiCompliance
-- |=================
```

```
odsiCompliance MODULE-COMPLIANCE
   STATUS      current
   DESCRIPTION
      "
      The implementation requirements for the ODSI MIB.
      "
   MODULE -- this module

      MANDATORY-GROUPS {
         odsiDeviceCommonGroup,
         odsiPortCommonGroup,
         odsiConnCommonGroup,
         odsiSigCommonGroup
      }

      GROUP odsiPortUserDeviceGroup
      DESCRIPTION
            "
            User devices are required to implement this group.
            "

      GROUP odsiPortEndPointUserDeviceGroup
      DESCRIPTION
            "
            User devices are required to implement this group.
            "

      GROUP odsiInBandChanCommonGroup
      DESCRIPTION
            "
            Devices that implement in-band control channels are required
            to implement this group.
            "

      GROUP odsiInBandChanSonetGroup
      DESCRIPTION
            "
            Devices that implement in-band control channels on SONET
            ports are required to implement this group.
            "

      GROUP odsiInBandChanDigitalWrapperGroup
      DESCRIPTION
            "
            Devices that implement in-band control channels on ports
            that employ digital wrappers are required to implement
            this group.
            "

      OBJECT      odsiPortLocalPortId
      MIN-ACCESS  read-only
      DESCRIPTION
         "read-only access is sufficient, if manual configuration of
          local port ID is not supported by the device.
         "

      OBJECT      odsiPortLocalMinChanSize
      MIN-ACCESS  read-only
      DESCRIPTION
         "read-only access is sufficient, if manual configuration of
          this parameter is not supported by the device.
         "

      OBJECT      odsiPortLocalConcatenation
```

```
      MIN-ACCESS  read-only
      DESCRIPTION
         "read-only access is sufficient, if manual configuration of
          this parameter is not supported by the device.
         "


   ::= { odsiCompliances 1 }

END
```

7. Acknowledgements

This document was produced by the ODSI Coalition.

8. Security Considerations

No known security considerations exist other than those imposed by SNMP itself.

9. References

[CRLDP]
Jamoussi, B., "Constraint-Based LSP Setup using LDP", draft-ietf-mpls-cr-ldp-03.txt, September 1999.

[LDP]
Andersson, L., Doolan, P., Feldman, N., Fredette, A., Thomas, B., "LDP Specification", draft-ietf-mpls-ldp-06.txt, October 1999.

[ODSIFS]
ODSI Coalition, "Optical Domain Service Interconnect (ODSI) Functional Specification", v1.2, April 2000.

[ODSIPPP]
ODSI Coalition, "ODSI Service Discovery and Address Registration", v1.1, April 2000.

[ODSISIG]
ODSI Coalition, "ODSI Signaling Control Specification", v 1.4, Oct 2000.

[RFC1155]
Rose, M., McCloghrie, K., "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[RFC1157]
Case, J., Fedor, M., Schoffstall, M., Davin, J., "A Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[RFC1212]
Rose, M., McCloghrie, K., "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[RFC1215]
Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[RFC1332]

McGregor, G., "The PPP Internet Protocol Control Protocol (IPCP)", RFC 1332, May 1992.

[RFC1661]
Simpson, W., "The Point-to-Point Protocol", STD 51, RFC 1661, July 1994.

[RFC1901]
Case, J., McCloghrie, K., Rose, M., Waldbusser, S., "Introduction to Community-based SNMPv2", January 1996.

[RFC1905]
Case, J., McCloghrie, K., Rose, M., Waldbusser, S., "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906]
Case J., McCloghrie, K., Rose, M., Waldbusser, S., "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2205]
Braden, R., Zhang, L., Berson, S., Herzog, S., Jamin, S., "Resource Reservation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[RFC2233]
McCloghrie, K., Kastenholz, F., "The Interfaces Group MIB using SMIv2", RFC 2233, November 1997.

[RFC2570]
Case, J., Mundy, R., Partain, D., Stewart, B., "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

[RFC2571]
Harrington, D., Presuhn, R., Wijnen, B., "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2572]
Case, J., Harrington, D., Presuhn, R., Wijnen, B., "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[RFC2573]
Levi, D., Meyer, P., Stewart, B., "SNMP Applications", RFC 2573, April 1999.

[RFC2574]
Blumenthal, U., Wijnen, B., "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", April 1999.

[RFC 2575]
Wijnen, B., Presuhn, R., McCloghrie, K., "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2578]
McCloghrie, K., Perkins, D., Schoenwalder, J., "Structure of Management Information Version 2", STD 58, RFC 2578, April 1999.

[RFC2579]
McCloghrie, K., Perkins, D., Schoenwaelder, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC2580]
McCloghrie, K., Perkins, D., Schoenwalder, J., "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[RFC2685]
Fox, B., Gleeson, B., "VPN Identifiers", RFC 2685, September 1999.

[RSVPRR]
Berger, L., Gan, D., Swallow, G., Pan, P., Tommasi, F., Molendini, S., "RSVP Refresh Overhead Reduction Extensions", draft-ietf-refresh-reduct-03.txt, March 2000.

10 Editor's Addresses

K. Arvind
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x124
Fax: 978-264-0671
Email: arvind@tenornetworks.com

Rob Coltun
Siara Systems
300 Ferguson Drive
Mountain View, CA 94043
Phone: (650) 390-9030
EMail: rcoltun@siara.com

John T. Moy
Sycamore Networks
10 Elizabeth Drive
Chelmsford, MA 01824
Phone: (978) 367-2161
Email: jmoy@sycamorenet.com

Arnold N. Sodder
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x110
Fax: 978-264-0671
Email: asodder@tenornetworks.com

Jeffrey Weiss
Ciena Access Systems Division
400 Nickerson Road
Marlborough, MA 01752
Phone: 508-486-3502
Email: jweiss@ciena.com

APPENDIX III

ODSI Coalition
August 2000
Version 1.4

G. Bernstein
J. Weiss
Ciena
R. Coltun
Siara Systems
J. Moy
Sycamore Networks
A. Sodder
K. Arvind
Tenor Networks
Editors

Optical Domain Service Interconnect (ODSI) Functional Specification

Foreword

The goal of ODSI is to allow user devices to dynamically request bandwidth from the Optical network. The Optical network consists of transmission and switching equipment that can provide point-to-point connections to attached user devices. These connections can be of various shapes and sizes, such as SONET-encoded leased line service, gigabit ethernet, even dedicated optical wavelengths where the encoding need not be known to the optical network itself. Typical user devices requesting these bandwidth services from the Optical network would include IP routers, SONET add/drop multiplexers, and ATM switches. Bandwidth received from the Optical network will then be used as point-to-point connections between routers, or trunks between add/drop muxes and ATM switches.

Current provisioning of bandwidth within the Optical network is basically static; a slow and painstaking operation that takes much configuration and often requires redesign of optical network internals. However, two simultaneous developments in networking technology make the dynamic request of bandwidth from the optical network both feasible and desirable. First, a new generation of optical switches is enabling dynamic point-and-click bandwidth provisioning by network operators. Second, traffic engineering [Ref5] and constraint-based routing enhancements to IP routers [Ref6, Ref7] and/or ATM switches are allowing these devices to dynamically determine when and where they need additional bandwidth. Allowing these devices to make requests to the Optical network directly reduces the workload on the network operator.

This document provides a technical overview of ODSI. Detailed protocol specifications can be found in a separate series of documents ([Ref11], [Ref12], [Ref13], and [Ref19]).

Table of Contents

1. Introduction

The Optical Domain Service Interconnect (ODSI) defines a interface to the Optical network whereby user devices, such as routers, SONET add/drop multiplexers, and ATM switches can dynamically request bandwidth from the optical network. This allows leased lines to be created (dialed) and destroyed in an on-demand fashion.

This is not a new concept. For example, creation of bandwidth in an on-demand fashion is currently possible through ISDN, ATM and Frame Relay Switch Virtual Circuits (SVCs), as well as the PSTN. The main difference between ODSI and these technologies is that ODSI is dealing with much larger bandwidth channels and is going to be used by user devices that are already aggregating a lot of traffic. As mentioned in the foreword, advances in technology are enabling the extension of bandwidth-on-demand concepts to the higher bandwidth optical network.

Since bandwidth-on-demand schemes have been around for years, there are a number of protocols which can be used to signal bandwidth requests. These protocols must be modified/enhanced to support characteristics unique to the Optical network. Since TCP/IP has become the universal control protocol for data communications, we choose to extend IP addressing and protocols to signal bandwidth requests to the optical network, as discussed in Sections 2 and 3. Complete specification of these protocol extensions will be the subject of a separate document.

ODSI is complementary to the proposal "Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control With Optical Crossconnects" [Ref4], which provides an outline of how to implement the internals of an Optical network. ODSI integrates the two levels of traffic engineering, that provided by the IP routers [Ref5] and the traffic engineering implemented in the optical network (one possible framework for which is presented in [Ref4]), while maintaining a separation that allows the two levels to evolve separately and protects one level from the other's failures.

1.1. Terminology

Throughout this document we use the following terminology. Note that some terms, such as "admission control" and "third-party", are used somewhat differently than their traditional sense.

- The endpoints at the end of the dynamic bandwidth connection are called "user devices". These can be IP routers, ATM and Frame Relay switches, SONET ADMs, etc.

- The collection of network devices creating the bandwidth connection are termed the "optical network". Individual components of the optical network are called "network devices". In particular, a user device connects to the optical network through a network device. Note that the optical network can contain non-optical components, such as traditional SONET cross-connects and ADMs. Likewise, some of the user devices may employ optical technology, such as IP routers with optical interfaces.

- The dynamically created bandwidth connecting two user devices is called a "bandwidth channel" or "circuit".

- "Third-party" signaling means simply that a user device (such as a network management station or provisioning server) can request that bandwidth be created between two user endpoints, neither of which is the requester itself. This does not imply that the endpoints themselves are incapable of signaling. In fact, in ODSI third-party signaling the two endpoints participate in signaling, they just don't initiate the request.

- By "admission control" control is meant the authorization function that the network performs to ensure that the requester is allowed to setup up a connection between two endpoints. This authorization is based on the user group that the requester and endpoints belong to, and by the Contract ID that is included in ODSI bandwidth requests. "Admission control" is not used to denote the function whereby the optical network determines that there are sufficient resources for the connection.

- A control channel available for ODSI messaging that is directly related to the user device's port into the optical network is called an "in-band control channel". These channels also go by the name facility-associated signaling in the literature; one example is the SONET Line Data Communication Channel (DCC) contained within an OC-48 port's SONET Line overhead bytes. Channels unrelated to any port into the optical network, such as a 10-base-T hub connecting the user and network device, are called "out-of-band control channels" (or non-facility associated signaling).

1.2. Document Organization

This document is organized into the following Sections:

- Section 2, "Service specification", provides a description of what can be requested from the optical network: how to identify the endpoints to be connected, and the characteristics of the bandwidth channel that will connect them.

- Section 3, "Protocols", gives a brief overview of the protocols that are used to make bandwidth-on-demand requests of the optical network.

- Section 4, "Example usage", gives a simple example of how the protocols are used to bring up dynamic leased line service between routers.

- Section 5, "Traffic engineering example", gives a more detailed example. In particular, this example shows how traffic engineering and constraint-based routing within the user devices attached to the optical network can be used to detect when and where to request additional bandwidth.

- Section 6, "Accounting/security", describes protocol options that enable bandwidth requests to be verified and logged, possibly for accounting or billing purposes.

- Section 7, "Management", describes the Management Information Base (MIB) that can be used to configure ODSI services and monitor the state of ODSI-related functions.

Various appendices provide ODSI information that is specific to various physical layer interfaces (e.g., SONET) over which ODSI functionality will be supported.

1.3. Revision History

Revision 1.4 of this document incorporated a number of changes to sychronize ODSI with the Optical UNI efforts taking place in the OIF and IETF [Ref17].

2. Service Definition

The service provided to the user devices attaching to the optical network is the dynamic creation of point-to-point bandwidth channels between user devices. There are three components to the service definition: 1) how do you specify the endpoints of the bandwidth channel 2) what are the characteristics of the bandwidth and 3) what actions can you request on the bandwidth channel. These components are expanded upon in the following subsections.

ODSI assumes the following model. A user device is connected to the optical network through one or more physical interfaces, such as OC-48 links. The user device wishes to dynamically connect these interfaces to other user devices attached to the optical network. When such a connection is established, it looks like a leased line connecting the two user devices. The optical network is providing a bandwidth service, but one that is not packet-based. The optical network has no concept of queueing or other packet-based quality of service features; these functions are left to the user devices. If the user device's interface supports channelization, parts of the interface may be connected to separate destinations; for example, each of say 4 STS-12cs contained in an OC-48 interface may be connected by ODSI to separate remote routers. While the network devices do not examine the data packets transmitted over the user device's interface to the optical network, the physical layer of the interface may support a separate in-band control channel that can be used by the packet-based ODSI protocols described in Section 3. One example of an in-band control channel is the SONET Line Data Communication Channel (DCC) contained within the SONET Line overhead bytes.

ODSI is based on IP protocols and addressing, and assumes that the control network used by the Optical network, and extended to user devices through the use of ODSI, is IP-based.

The IP addresses assigned to the user devices serve multiple purposes: as endpoint IDs of circuits, and as the source and destination of ODSI protocol packets. The latter function implies that the IP addresses are routable, at least in the context of the Optical network's IP control network.

2.1. Endpoint identification

Each endpoint of the requested bandwidth is specified as a combination of one or more of the following parameters.

- IP address. An IP address is associated with a user device, and can be associated with one, some, or all of that device's physical port attachments to the optical network.
- Port index. Identifies a particular physical interface connecting into the Optical network. In IP MIBs, this is commonly an IfIndex.
- Channel. When establishing sub-rate bandwidth connections on multiplexed interfaces, this identifies which sub-rate channel should be connected. For example, on a SONET interface this would be a timeslot, as specified in Section

6.1.2 of [Ref10].

- Container. This parameter further subdivides the channel. As an example, when the port uses SONET framing, "container" can be used to identify a virtual tributary within a channel. The encoding of both the channel and container identifiers depends on the port's physical layer framing.

- User group. A further qualification on the IP address; a port is only allowed to attach to IP addresses associated with the same user group, similar to the closed user group concept in X.25. For example, if there were 10 ISPs whose routers were all attached to the same optical network, each ISP would have a separate user group identifier. The User Group ID is encoded as a VPN identifier [Ref18]. See Section 6 for further details.

The only required parameters when requesting bandwidth are the IP addresses and user group of the two endpoints. When port/channel/container are not explicitly specified, any port associated with the IP address and matching the requested bandwidth characteristics will be connected.

2.2. Bandwidth characteristics

The characteristics of the dynamic bandwidth channel connecting two user devices can be specified by the following parameters:

- Framing. This parameter indicates the format of the signal on the port between user and network devices. It is defined to be one of:

  - PDH
  - SONET
  - SDH
  - Digital wrapper
  - LAN ethernet
  - WAN ethernet. This is transported in an OC-192c.

  Notes: (1) The same amount of bandwidth may be framed in different ways, (2) framing need not be explicitly mentioned if implied by the circuit endpoints, and (3) if supported by the network, framing at the two endpoints may be different.

- Transparency,. This parameter is interpreted with respect to the framing. It indicates the portion of the signal that is available for user data traffic. For SONET/SDH framing, the transparency options are:

  - PLR-C. Physical layer regeneration, meaning all the SONET signal is available for user data.

  - STE-C. Endpoints are attached to section-terminating equipment. Only line and path overhead are available to user data traffic.

  - LTE-C. Endpoints are attached to line-terminating equipment. Only path overhead is available to user data traffic.

- Size. Examples include STS-1, OC-48, and 1 or 10 megabit (ethernet). Specified in bits/second. Any size bandwidth may be requested, although it is not required that all sizes be supported by the optical network. When a size is requested that the optical network does not support, it may return a larger bandwidth connection, but never a smaller. However, if the amount of bandwidth requested is not available, the optical network will indicate the maximum amount of bandwidth that can be obtained to the specified endpoint.

- Service level. Service level encompasses priority (whether the circuit can be preempted by other, higher priority circuits), protection (whether the bandwidth should be protected against failures, and if so, the speed at which the protection will restore bandwidth service after a failure), and availabilty. Service level is encoded as an integer, whose meaning is assigned by the optical network provider. For example, one provider could encode its "gold service" as service level 1, which might be advertised by the provider as "50 millisecond restoral through BLSR technology".

- Propagation delay. The user device can specify the maximum amount of propagation delay to be tolerated on the circuit. Specified in milliseconds.

- Diversity. The user device may request that the new circuit be diverse (that is, share no common facilities) from an existing set of ODSI-created circuits.

  The encoding of ODSI's "diversity" bandwidth characteristic is defined as a list of one or more existing circuits,

specifying for each circuit:

- The circuit's identifier, which is the combination of an IP address of one of the circuit endpoints and a 32-bit ID assigned by the Optical network.

- The kind of diversity required from this existing circuit, one of (a) link diverse, (b) node diverse, or (c) SRLG diversity.

o Directionality. This bandwidth characteristic allows for both unidirectional and bidirectional circuits. In addition, this allows the specification of asymmetric circuits, by specifying such circuits as two unidirectional halves.

o Contract ID. This parameter provides billing and authorization information for the circuit. Contract ID is encoded as a string, and is assigned by the provider of the Optical network. It is included in bandwidth requests, and in that way is passed to the Optical network so that it can verify whether the requester is allowed to set up a particular circuit (e.g., does the requester have enough credit in their account).

o Vendor extensions. Vendors may create their own proprietary bandwidth descriptions. Mechanisms will be provided so that vendor extensions can be expressed during service discovery (Section 3.1) and by ODSI signaling (Section 3.3).

Some network operators may support a smaller set of options for ODSI bandwidth requests. For example, a network operator may decide to offer only OC-48 service, or limit availability to two choices: 4 hours per year or 2 days per year.

Details concerning particular physical layers, for example how ODSI encodes channel identifiers, or how ODSI protection mechanisms interact with protection schemes already defined for the physical layer, are defined in Appendices.

Framing, transparency and bandwidth size may be left unspecified when implied by endpoints of the bandwidth channel.

2.3. Actions

There are three actions that you can request regarding allocation of dynamic bandwidth. All actions are performed by a signaling protocol, as outlined in Section 3.3.

- Create (dial). Create a point-to-point bandwidth channel between two user devices connected to the optical network. The two endpoints of the channel and the characteristics of the bandwidth must be specified, as described in Sections 2.1 and 2.2.

- Destroy (hang-up). Destroy an existing bandwidth channel. Either one of the endpoints can be specified.

- Query (caller ID). Query the status of an existing bandwidth channel. Either one of the endpoints is specified, and the status (established or not) and other endpoint of the channel is returned.

- Modify. Increase or decrease the amount of bandwidth associated with actively connected circuit. The user device may also request to modify other bandwidth parameters.

- Non-destructive modify. Identical to the modify request, except for the following provision. For the network to successfully execute a non-destructive modify request, the requested changes must be performed in a hitless fashion: the existing bandwidth connection must continue to be presented to the user endpoints without errors.

- Directory lookup. Obtain a list of endpoints to which the user device can establish circuits. Each endpoint is identified by an IP address. Only IP addresses registered with the same user group as the device making the directory lookup will be returned. In addition, the device can restrict the lookup to only those endpoints supporting a specified physical layer framing.

Note the difference between a modify request and a destroy followed by a create with new parameters. If the former fails, the user device still has a connection. If the latter fails, the user device has no connection.

2.4. Unsupported services

The goal of ODSI is to provide an interface to the optical network that can be quickly implemented and deployed. We have chosen to limit the feature set accordingly, eliminating features whose utility is not yet obvious, or which can be derived from other features that are included by ODSI. In particular, the following features are not supported.

- Point-to-multipoint bandwidth channels. Routers and ATM switches will provide the multicast service on top of the point-to-point bandwidth provided by the Optical network. We are not going to try to specify the equivalent of a LAN emulation service at the Optical layer.

- VPN services. These too will be provided by the routers and ATM switches connected to the Optical network. The IP addresses which specify the endpoints of the dynamically created bandwidth channels must be routable within the optical network's control network. (Note: ODSI user groups provide some VPN functions by restricting which endpoints can be connected. However, unlike VPNs all ODSI user devices connected to a single optical network must use a single routable IP address space).

- Connections based on time of day. These services can be built upon the basic ODSI services outlined in Section 2, as vendor value-added features.

- NNI connections. These come in various degrees. ODSI does not support the connection between two elements internal to the optical network; instead, technology as outlined in [Ref4] would be more appropriate. However, ODSI could be used at the connection between two Optical networks. ODSI signaling would transit the boundary between two optical networks unaltered. You would also need to exchange endpoint identification. Since we are identifying endpoints by IP addresses, this could be accomplished in a limited scope by existing ODSI registration mechanisms, although a future extension to an IP routing protocol (such as BGP) would be better in the long term, as it would enable arbitrary inter-connection topologies.

- Diagnostic and test access. Examples include the ability to trace the path of an ODSI-supplied circuit, and test access to a circuit through a bridging capability. While not part of the ODSI specification, this support may be supplied through vendor extensions.

3. Protocols

The protocols involved in ODSI perform three functions. First, they allow a user device to detect that one of its ports is connected to an optical network that is capable of dynamic bandwidth allocation; we call this "service discovery". Second, they allow the user device to give an address to the port, so that the port can be used as an endpoint in a dynamic bandwidth channel creation request; we call

this function "address registration". Lastly, there is a "signaling" function, which the user device uses to dynamically connect its ports to other ports connecting to the optical network.

ODSI is implemented using IP protocols. ODSI protocol packets are transmitted over the IP control network composed of the optical network's network devices, the user devices and selected links interconnecting the two. These links are classified as either in-band, which use part of the same physical port that has been registered with ODSI for dynamic bandwidth creation, or out-of-band, which use entirely separate physical paths. SONET DCC overhead bytes were the in-band example mentioned previously; an example of an out-of-band control connection would be a 10-base-T ethernet hub connecting a user device's management ethernet to a network device's management ethernet interface.

Service discovery and address registration are accomplished via a PPP session running in an in-band channel associated with the port. This in-band channel should be available even if user traffic is present. For example, the port may have been initially connected via ODSI, or by other network management means, and then the user device restarted. The user device would still like to know that ODSI is available on the port, in case it is desirable to reconnect that port to a different destination at some later time. If an in-band channel is not available, information that would have been derived from service discovery and address registration must be configured in the user devices and their directly attached network devices.

Signaling messages are sent over the IP control network, possibly traveling multiple IP hops over both in-band and out-of-band control links.

The three protocol functions are described in more detail in the following subsections.

3.1. Service discovery

When a physical port of a user device connects to an Optical network, the device wishes to know whether it can use ODSI to dynamically connect the port, through the optical network, to another port on a remote device. This discovery is done using the PPP protocol [Ref1] over the in-band channel; the termination points for the PPP session being the user device and the network device to which it directly connects.

The user and network devices should periodically test the PPP connection for liveness, using PPP's Link Quality Monitoring Protocol [Ref14]. If the PPP connection fails, any address

registrations (Section 3.2) for the associated port are removed, but any existing bandwidth connections to the port are maintained.

The following pieces of information are exchanged over the port's in-band control channel during service discovery.

- The user device indicates the port's number, which is used to specify that this port be used for dynamically created bandwidth.

- The user device indicates the port's user group, used to restrict incoming connections to the port and to identify the user device for accounting purposes. The user device may need to provide a digital signature with the user group, so that its membership can be verified (see Section 6).

- The network device indicates that the port is available for dynamic bandwidth creation.

- The network device indicates its corresponding port number, which can be alternatively specified during bandwidth requests (see Appendix C).

- The network device provides its IP address, which can be used in combination with the network device's port number as an alternative endpoint identification.

- The network device provides the address of an "Optical networking controller", to which signaling requests can be addressed.

- Any physical characteristics of the port which cannot be dynamically determined are exchanged. For example, the user device can indicate whether it is capable of splitting the port into separate channels, and the network device can indicate whether the optical network is capable of routing these channels separately.

- The network device tells the user device whether address registrations and signaling requests must be authenticated (see Section 6).

- The network device indicates to the user device the typical bandwidth connection setup time, in milliseconds. The user device should treat this as advice as to whether it should set up a connection, and when it should give up on a pending connection.

3.2. Address registration

Address registration is a port-specific process, whereby the user device associates one or more IP addresses to the port; these IP addresses can then be used as endpoint identification in dynamic bandwidth connection requests (see Section 2.1). A user group is associated with each registered address. Address registration also uses PPP running over the port's in-band control channel.

A user device can advertise the same IP address for multiple ports. In this case, if a specific port is not specified in signaling requests for the advertised address, the optical network is allowed to pick any one of the associated ports.

The user device can also advertise multiple addresses for a single port. This capability is useful when implementing a simple NNI function (see Section 2.4). In this case, some (or all) of the addresses are being advertised by proxy for other (remote) user devices.

3.3. Signaling

The signaling protocol is used to request connection of two ports (or even subchannels of ports), disconnect a port, query the connection status of a port, modify the bandwidth of any existing connection or obtain a list of valid connection endpoints. Data elements carried by the signaling include identification of the endpoints to be connected (Section 2.1), and the bandwidth characteristics of the connection (Section 2.2).

Signaling is intrinsically third-party: neither of the connection endpoints need belong to the requesting user device. When the optical network receives the connection request, it performs the following functions:

- The optical network validates the connection request.
- The network determines whether a path exists for the connection that meets the requested bandwidth constraints.
- The network informs the two endpoints of the pending connection, and upon their acceptance sets up the bandwidth and informs the requester of the result.

Signaling requests are IP messages. They can be forwarded on any IP data path between the requesting user device and one of the

network devices. This could be through an in-band control channel. Or it could be through connections that were reserved for control traffic - for example, a 10-base T ethernet connecting the user device and the network device.

The identification of the endpoints may be only partially specified. In this case, the optical network will choose appropriate endpoints, and this choice will be conveyed to the endpoint devices before the bandwidth is fully established.

Once a bandwidth connection has been established, it can be identified in signaling messages by either a) a complete specification of either endpoint, down to the channel number or b) a combination of one endpoint's IP address and a connection ID assigned by the signaling protocol.

The following points should be kept in mind concerning ODSI signaling:

- ODSI signaling is intrinsically third-party. Bandwidth can be requested by a device other than the endpoints.
- ODSI signaling messages need not follow the same path as the bandwidth connection that will be established.
- The endpoints should be able to influence the connection, including refusing the connection, or specifying particular ports.
- The implementation of the optical network is unspecified; it can be either distributed or centralized circuit placement algorithms.
- User and network devices are part of a single IP network for control purposes.
- Devices are not required to support the MIB. In particular, SNMP sets cannot be used as a replacement for signaling.
- Unlike MPLS, in ODSI there are no labels to set up. The resulting bandwidth connection will look like a leased line (or full duplex gigabit ethernet, etc.) to the user devices, and will be space-division switched, as opposed to packet-switched, by the optical network. (The endpoint or channel number could be considered the equivalent of an MPLS label, but does not appear anywhere in the data stream).

4. Example usage

This section presents a simple example of ODSI functionality. Three IP routers, A, B, and C are attached to an optical network through three Optical switches SW1, SW2 and SW3, via OC-48 SONET links, as pictured in Figure 1. The goal of ODSI is to allow Router A to dynamically choose whether it is connected to Router B or C (or neither). When connected, the connection will be essentially a leased line between the routers.

Assume that Router A can support an in-band control channel on its OC-48, but that B and C cannot. Router A can dynamically discover that ODSI is available on its OC-48 link into the optical network, can tell SW1 its IP address, and can learn SW1's address, which will be the destination of any signaling requests. In contrast, these items must be configured in Routers B and C and their attached network devices SW2 and SW3. In addition, in order to participate in signaling, an out-of-band control connection must exist between the latter two routers and their network devices, pictured in the diagram as 10baseT ethernet segments.

If Router A now wants a leased line to B, it sends an ODSI signaling request to SW1 over the in-band channel, requesting a SONET OC-48 connection to Router B's IP address. The Optical network then verifies that such a connection can be established to B, confirms

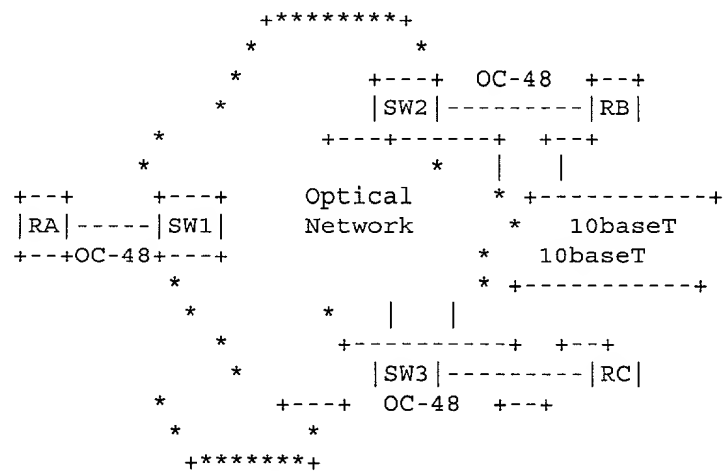


Figure 1: Simple ODSI example

with B that this is OK (over the 10baseT), sets up the connection and returns a success indication to A.

Two questions come up in this example. How does Router A know Router B's IP address, and how does it know that it wants to connect to Router B. Router A may have some a priori knowledge (such as an operator configuring the information), but as we shall see in the next section, a bootstrapping scheme using MPLS traffic engineering may also supply the needed information.

5. Traffic engineering example

Figure 2 shows a sample optical network of three optical switches, interconnected by WDM links and providing leased line services for four IP routers.

Assume that OC-48 leased lines have been provisioned between the routers, using traditional means, producing a linear topology amongst the routers as shown in Figure 3. This allows IP connectivity to be established between the routers, enabling them to learn each others' IP addresses.

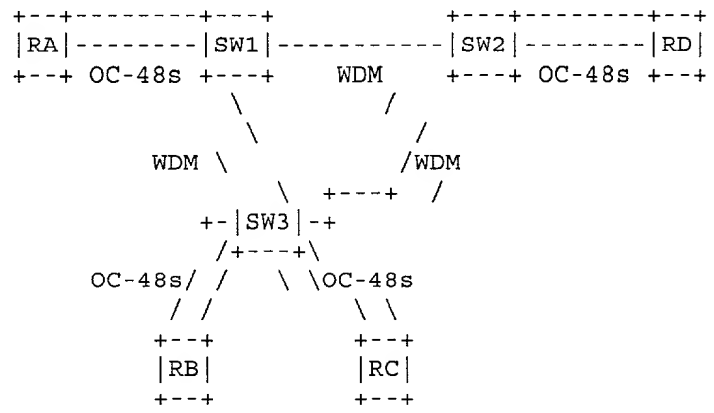


Figure 2: Simple Traffic Engineering example

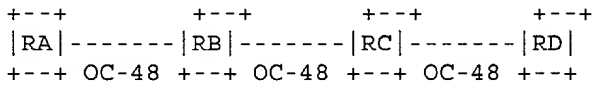


Figure 3: Bootstrap router topology

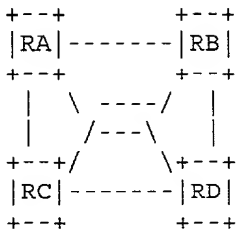


Figure 4: LSP overlay (STS-12c LSP size)

On top of this, a combination of MPLS and constraint-based routing allows you to overlay a full mesh of Label Switched Paths (LSPs) between routers for traffic engineering purposes, as shown in Figure 4. Assume at first that it is estimated that the traffic between each pair of routers is an STS-12c (i.e., each LSP is sized at STS-12c). Then the router topology in Figure 3 suffices. However, if the estimate of the IP traffic between Routers A and D increases to 2 OC-12s, constraint-based routing will tell the routers that the LSPs no longer fit. Dialing up a new OC-48 link through ODSI between A and D (or B and C) fixes the problem.

[Editors' note: The routers in this example could also bootstrap themselves by using the ODSI "Directory lookup" to find the other routers connecting to the Optical network.]

6. Accounting/security

The ODSI security model uses both Contract IDs and user group IDs. A Contract ID is encoded as a string, and is assigned by the provider of the Optical network. It is included in signalling messages by the Requester, and in that way is passed to the Optical network so that it can verify whether the requester is allowed to set up a particular circuit (e.g., does the requester have enough credit in their account). The network must be able to verify the Contract ID, and so it is signed when appearing in ODSI messages.

The User Group ID is encoded as a VPN identifier [Ref3]. It too is included by the requester in signalling messages, but it is administered by the user devices, and not the Optical network. User devices use the user group ID to verify circuit add/delete/modify requests. Verification of user group membership is optional. When user group membership is to be authenticated, address registrations and signaling requests carry digital signatures in order to prove membership.

Bandwidth will only be established between user devices that can be verified to belong to a common user group. The requester of the bandwidth must also belong to the same user group. Note however that a single device may belong to multiple user groups, as may be desirable for a provisioning server used by multiple user communities.

Admission control and accounting of bandwidth requests is accomplished through extensions to the COPS protocol [Ref9], [Ref19].

7. Network management

A Management Information Base will be created for ODSI, consisting of the following tables. These tables are present in both the user and network devices.

- A port table, containing the user group identifier, port ID, network device port ID, network device IP address, and a pointer to a table for physical layer parameters (which may include information concerning channelization, in-band control, and so on).
- Port address table. The list of addresses that have been registered for a port, and their user groups.
- Connection table. Information about current connections. Indexed by the connection identifiers in Section 2.2.
- Available endpoint table. The list of endpoints that are available in a given user group.

Appendix A. SONET-specific ODSI attributes

This Appendix provides ODSI information specific to the SONET physical layer.

A.1 Channel numbering

The fundamental signal in SONET is the STS-1 (about 51 Mbps). This signal consists of a transport overhead and a Synchronous Payload Envelope (SPE). The SPE floats within its allotted space within the SONET frame structure with the pointer bytes (H1, H2 and H3) in the Line Overhead of the SONET transport overhead pointing to the beginning of the SPE. An STS-N signal is formed from a SONET STS-(N- 1) signal and an STS-1 signal via byte interleaving. The transport overhead structures are frame aligned prior to interleaving but this is not required of the SPEs, i.e., there is no special relationship between the payload envelopes. To transport signals in excess of about 50Mbps the SPEs can be concatenated, i.e., glued together. In this case their relationship with respect to each other is fixed in time and hence this relieves, when possible, and end system of any inverse multiplexing bonding processes.

The end points of SONET connections can be identified by the "time slots" (position) that they occupy within the interleaved frame structure. In the standard SONET case you must specify which of the M STS-1 paths within an STS-N signal will be used to transport the data (M <= N, and N = 3, 12, 48, 192,...). The SPEs of these M STS-1s can be concatenated to form an STS-Mc. The STS-Mc notation is really a short hand way of describing an STS-M signal whose SPEs have been concatenated.

In BellCore GR-253 [Ref10] section 6.1.2 (requirement R6-3) two conventions are given for identifying an STS-1 within an STS-N:

- two-level "STS-3 #, STS-1 #"
- A single-level "1 to N in order of appearance at the input to the byte-interleaver"

For example STS-1 number 23 within an OC-48 can also be represented by the tuple (8, 2). ODSI uses the single-level numbering scheme to identify endpoints.

A second complication is in dealing with concatenated signals. In Bellcore GR-253 section 5.1 the multiplexing procedures for SONET are given. Constraints are imposed on the size of STS-Mc signals, i.e., they must be a multiple of 3, and on their

starting location and interleaving. This has the following advantages: (a) restriction to multiples of 3 helps with SDH compatibility (there is no STS-1 equivalent signal in SDH, an STM-1 is equivalent (essentially) to an STS-3c); (b) the restriction to multiples of 3 reduces the number of connection types; (c) the restriction on the placement and interleaving could allow more compact representation of the "label"; The major disadvantage of these restrictions are: (a) Limited flexibility in bandwidth assignment (somewhat inhibits finer grained traffic engineering). (b) The lack of flexibility in starting time slots for STS-Mc signals and in their interleaving (where the rest of the signal gets put in terms of STS-1 slot numbers) leads to the requirement for re-grooming (due to bandwidth fragmentation).

Due to these disadvantages ODSI will support "arbitrary" concatenation, i.e., no restrictions on the size of an STS-Mc (as long as M <= N) and no constraints on the STS-1 timeslots used to convey it. However, the use of arbitrary concatenation or "standard" concatenation will be negotiated as part of ODSI service discovery process.

A.2 In-band control channels

SONET currently references three types of spans, Section, Line and Path. A section covers the span between two intelligent repeaters (such as OEO regenerators) or between terminating equipment and an intelligent repeater, or between terminating equipment for which there are no intelligent repeaters. A "Line" covers the span between two pieces of terminating equipment such as SONET Add Drop Multiplexers (ADMs) or Optical Cross Connects (OXCs) supporting Optical to Electrical to Optical Conversion (OEO). A "Path" covers the end to end span through a SONET network and terminates on both end points, for example IP routers. This concept is illustrated in Figure 5.

Telcordia and ANSI have defined specific overhead bytes which are to be terminated and processed at network elements corresponding to these designations.

The frame structure of SONET allocates approximately 3% of the framing format for maintenance purposes. In particular, in each 810 byte STS-1 frame, 27 bytes are reserved for section and line maintenance. An additional 9 bytes are reserved for path maintenance on a per concatenated payload basis. See ITU-T G.709 or Telcordia GR-253 for additional information on framing formats.

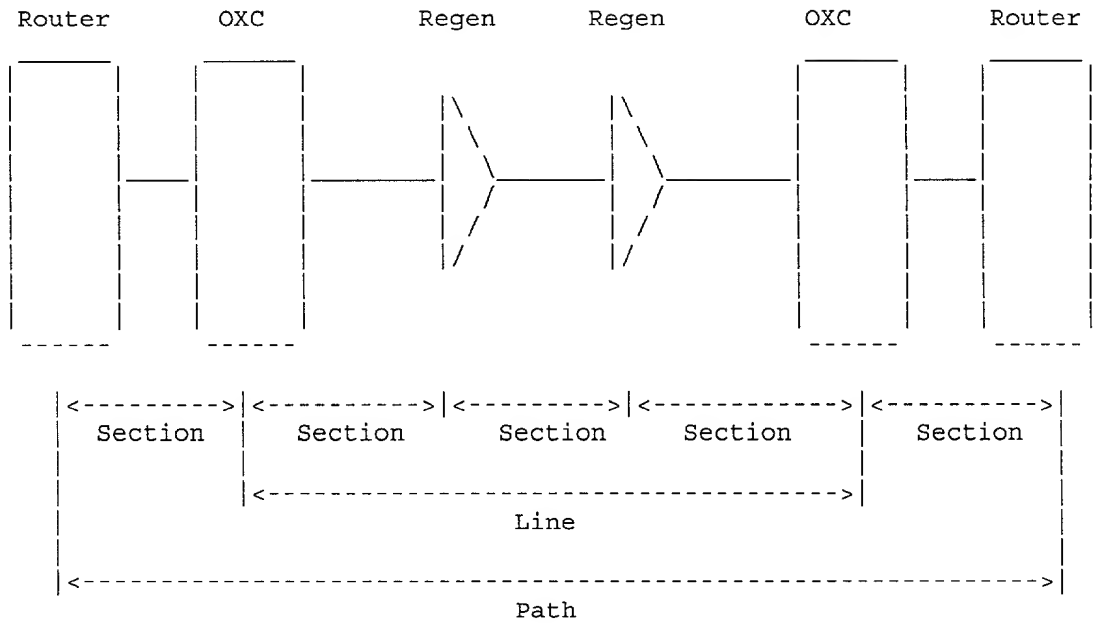


Figure 5: SONET span types

The Section, Line and Path Overhead bytes are labeled as follows:

tab(;),center; l l. Name;Description _
A1,A2;Section Framing bytes B1;Section BIP-8 error monitoring for previous payload B2;Line BIP-8 for each STS-1 individually (BIP is a ;parity computation) B3;Path BIP-8 over payload C1;STS-1 Identifier (fixed value) C2;Signal Label, indicates construction of STS Nc ATM D1,D2,D3;Section network management channel 192 Kbps ;(Section DCC) D4->D12;Line network management channel, 576 Kbps ;(Line DCC) E1;Section orderwire, used for mux to mux handsets E2;Line orderwire F1;Section User defined F2;User Channel G1;Path FEBE, RDI, Loss of Cell delineation for ATM H1,H2,H3;Line structure pointer and pointer action H4;Indicator for various SONET boundaries J0;Section Trace-J0 J1;Path Trace, provides connectivity verification K1,K2;Line automatic protection switching. K2 also ;carries line AIS, line RDI, RDI removal M0 or M1/Z2; Line Far end block error count (FEBE). ;Reflects received errors back to source. S1;Line Sync Status Indicates quality of clock source Z0;Section Growth Z1,Z2,Z3,Z4;Future

Each overhead byte position utilizes 64 kbps bandwidth

SONET byte interleaves multiple STS-1s to achieve higher data rates. For SONET data streams greater than a single STS-1, the Section, and Line overhead for the first STS-1 is processed, and the overhead of subsequent STS-1s is predominantly unused except for framing and parity..

Most commercial hardware framers support termination of Section (D1,D2 and D3 - 192 Kbps) and Line (D4->D12 576 Kbps) HDLC communications channels. This implies that most existing router and ADM interfaces are already able to communicate on these channels. The GR253 specified control channel between adjacent SONET Network Elements is the section DCC. To minimize interference with existing SONET NE control, this specification recommends the use of Line DCC as the default for the in-band control channel, unless administratively configured to use alternative overhead bytes.

SONET overhead bytes can be used to provide an in-band control channel for ODSI service discovery, address registration and signaling. The default in-band control channel for an ODSI interface will occupy the SONET Line Overhead bytes labeled D4, D5 through D12 of the first STS-1 of the SONET structure. (576 Kbps). The configured overhead bytes are then concatenated to form a synchronous, bit-stuffed HDLC-framed stream, over which the PPP protocol is then run.

It is recommended that both user devices and network devices allow the configuration of an in-band control channel to comprise one or more alternative overhead bytes: D1, D2, D3, E1, E2, F1, F2, Z1, Z2, Z3, Z4.

A.3 Detection of established connections

When the SONET port has not yet been connected via ODSI to a remote endpoint, a SONET AIS pattern will be received on the port. Circuit failures can also be detected via standard SONET mechanisms.

A.4 Interaction with SONET protection schemes

Standard SONET protection schemes can be used between the user devices and their attached network devices, providing local protection of a port and its ODSI connection. For example, ports A and B of a user device can be connected to a network device, and standard 1:1 protection can be configured with port A working and port B protect. Separate ODSI connections can then

be established on ports A and B. However, if A fails its ODSI connection will be rerouted to B, replacing B's previous connection. (This naturally gives port B's traffic lower priority, which is in accordance with the "extra data" traffic on rings and linear APS per Bellcore, ANSI and ITU specifications.)

Appendix B. Digital wrappers

This Appendix provides ODSI information specific to physical links employing Digital wrapper framing. If the wrapped information is SONET, most of the information in Appendix A applies. However, Digital wrappers employ a different in-band control channel.

B.1 In-band control channels

The optical channel overhead (sometimes referred to as the user data channel) can be used to provide an in-band control channel for ODSI service discovery, address registration and signaling. There are 40Mb/sec of overhead bytes available for this purpose, which can be viewed as four 10Mb/sec channels. Both user devices and network devices should allow the configuration of one of these channels as an in-band control channel, which should be encoded using 10Mb full-duplex ethernet framing, over which the PPP protocol is then run.

Appendix C. User devices without IP addresses

All user devices which implement ODSI need IP addresses, so that they can exchange ODSI protocol packets, which are encapsulated within the IP protocol. However, the network devices can dynamically assign IP addresses to a user device, using the standard IP Control Protocol [Ref2] of PPP, running over a port's in-band control channel.

The user device would use this address as the source address in its signaling packets, but it wouldn't want to use the address as the endpoint of a bandwidth connection request since it's been dynamically allocated from a temporary pool. Instead, the network device's IP address and port number can be used as the endpoint of connection requests.

Appendix D. SONET ADMs

SONET Add/Drop multiplexers can be either user or network devices. When they are user devices, they cannot request specific SONET protection modes such as ring or Linear APS for their dynamic bandwidth connections. Instead, they are forced to build their own protection using ODSI primitives as building blocks: allocating sets of diverse connections to build their own ring and linear protection schemes.

References

[Ref1] Simpson, W., "The Point-to-Point Protocol (PPP)", RFC 1661, July 1994.

[Ref2] McGregor, G., "The PPP Internet Protocol Control Protocol (IPCP)", RFC 1332, May 1992.

[Ref3] Rosen, E., A. Viswanathan and R. Callon, "Multiprotocol Label Switching Architecture", work in progress.

[Ref4] Awduche, D., Y. Rekhter, J. Drake and R. Coltun, "Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control With Optical Crossconnects", work in progress, November 1999.

[Ref5] Awduche, D., J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, "Requirements for Traffic Engineering Over MPLS", RFC 2702, September 1999.

[Ref6] Smit, H., T. Li, "IS-IS extensions for Traffic Engineering", work in progress, May 1999.

[Ref7] Katz, D. and D. Yeung, "Traffic Engineering Extensions to OSPF", work in progress.

[Ref8] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[Ref9] Boyle, J., R. Cohen, D. Durham, S. Herzog, R. Rajan and A. Sastry, "The COPS (Common Open Policy Service) Protocol", work in progress, November 6, 1999.

[Ref10] Bellcore Generic Requirements, GR-253-CORE, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, Issue 2, December 1995.

[Ref11] ODSI Coalition, "ODSI Service Discovery and Address Registration", work in progress.

[Ref12] ODSI Coalition, "Definition of Managed Objects for ODSI Management", work in progress.

[Ref13] ODSI Coalition, "Optical Domain Service Interconnect (ODSI) Signaling Specification", work in progress.

[Ref14] Simpson, W., "PPP Link Quality Monitoring", RFC 1989, August 1996.

[Ref15] Wright A. and R. Barry, "Proposed Architecture Working Group Requirements Document (Project 1)", OIF00.007.3, January 31, 2000.

[Ref16] Bernstein, G. and T. Tedijanto, "UNI requirements for topology discovery", OIF99.160, December 3, 1999.

[Ref17] Aboul-Magd, Osama, et al., "Signaling Requirements at the Optical UNI", work in progress.

[Ref18] Fox B. and B. Gleeson, "VPN Identifiers," RFC 2685, Sepember 1999.

[Ref19] ODSI Coalition, "COPS Usage for ODSI", work in progress.

Editors' Addresses

Greg Bernstein
Ciena Corporation, Core Switching Division
10201 Bubb Road
Cupertino, CA 95014
Email: greg@ciena.com

Jeffrey Weiss
Ciena Access Systems Division
400 Nickerson Road
Marlborough Mass 01752
Phone: 508 486-3502
Email: jweiss@ciena.com

Rob Coltun
Siara Systems
300 Ferguson Drive
Mountain View, CA 94043
Phone: (650) 390-9030
EMail: rcoltun@siara.com

John T. Moy
Sycamore Networks
10 Elizabeth Drive
Chelmsford, MA 01824
Phone: (978) 367-2161
Email: jmoy@sycamorenet.com

Arnold N. Sodder
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x110
Fax: 978-264-0671
Email: asodder@tenornetworks.com

K. Arvind
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x124
Fax: 978-264-0671
Email: arvind@tenornetworks.com

APPENDIX IV

ODSI Coalition
April 2000
Version 1.1

G. Bernstein
Ciena
R. Coltun
Siara Systems
J. Moy
Sycamore Networks
A. Sodder
K. Arvind
Tenor Networks
Editors

ODSI Service Discovery and Address Registration

Abstract

ODSI is a suite of protocols allowing internetworking devices to dynamically request bandwidth from the Optical network. Part of the ODSI protocol suite performs service discovery and endpoint registration, and is based on the PPP protocol.

The Point-to-Point Protocol (PPP) [Ref2] provides a standard method of encapsulating Network Layer protocol information over point-to-point links. PPP also defines an extensible Link Control Protocol, and proposes a family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

This document defines a method of exchanging ODSI information over a PPP link. The link is assumed to be an in-band control channel within a larger optical connection to the Optical network. The ODSI information allows each device to discover the ODSI parameters related to the optical connection; for example, is ODSI available, what is the address to which bandwidth connection requests should be sent, and what are the endpoint identifiers to be associated with the optical connection.

Table of Contents

1. Introduction

PPP has three main components:

(1) A method for encapsulating datagrams over serial links.

(2) A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection.

(3) A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

In order to establish communications over a point-to-point link, each end of the PPP link must first send LCP packets to configure and test the data link. After the link has been established and optional facilities have been negotiated as needed by the LCP, PPP must send NCP packets to choose and configure one or more network-layer protocols. Once each of the chosen network-layer protocols has been configured, datagrams from each network-layer protocol can be sent over the link.

This document describes how ODSI service discovery and address registration are performed over PPP, including the encapsulation of ODSI messages performing these functions. The negotiation of ODSI usage on the optical connection and the various parameters affecting ODSI operation is performed using the ODSI Control Protocol (ODSICP).

2. ODSI Control Protocol

The ODSI Control protocol (ODSICP) is responsible for ODSI service discovery and endpoint registration for a connection between an internetworking device and the optical network. ODSICP runs over an in-band channel contained within the optical connection. The exact configuration of the in-band channel depends upon the physical framing of the optical connection; for example, on a SONET connection certain SONET overhead bytes are concatenated into a HDLC bit-oriented stream [Ref1].

ODSICP uses the same packet exchange mechanism as the Link Control Protocol (LCP). ODSICP packets may not be exchanged until PPP has reached the Network-Layer Protocol phase. ODSICP packets received before this phase is reached should be silently discarded.

The ODSI Control Protocol is exactly the same as the Link Control Protocol [Ref2] with the following exceptions:

Data Link Layer Protocol Field

Exactly one ODSICP packet is encapsulated in the Information field of PPP Data Link Layer frames where the Protocol field indicates type hex TBD (ODSI Control Protocol).

Code field

Only Codes 1 through 7 (Configure-Request, Configure-Ack, Configure-Nak, Configure-Reject, Terminate-Request, Terminate-Ack and Code-Reject) are used. In addition, four new codes have been defined for ODSI: Register-Address, Withdraw-Address, Register-Ack and Register-Nak. All other Codes should be treated as unrecognized and should result in Code-Rejects.

Timeouts

ODSICP packets may not be exchanged until PPP has reached the

Network-Layer Protocol phase. An implementation should be prepared to wait for Authentication and Link Quality Determination to finish before timing out waiting for a Configure-Ack or other response. It is suggested that an implementation give up only after user intervention or a configurable amount of time.

Configuration Option Types

ODSICP has a distinct set of Configuration Options, which are defined in Section 2.2 of this memo.

2.1. Theory of operation

The use of ODSICP for ODSI service discovery and address registration (also called endpoint registration) proceeds as follows. Throughout we assume that PPP is run on an in-band channel of a particular connection to the Optical network.

(1) The use of ODSI on the optical connection is negotiated using Configure-Request and Configure-Ack. ODSI is in use on the Optical connection if and only if the ODSICP is in Opened state.

(2) When the ODSICP protocol first reaches Opened state, there are no addresses registered with the connection. While ODSICP is in Opened state, address can be associated with the optical connection using Register-Address. Previously associated addresses can be removed by Withdraw-Address.

(3) If the ODSICP is closed (through use of Terminate-Request or other means) all addresses registered for the optical connection are immediately withdrawn. However, current bandwidth connections are not severed, but remain in force.

(4) Unlike most other PPP Control Protocols, ODSI does not have a separate datagram channel, and needs only a control protocol assignment. (Contrast with IPCP, which uses PPP protocol hex 0021 for IP data packets and PPP protocol hex 8021 for IPCP control packets). ODSI signalling packets can be sent over the PPP link, but are encapsulated as IP packets and so use PPP protocol hex 0021.

(5) If the device connecting to the Optical network is relying upon IPCP to assign it an IP address, IPCP must run before ODSICP, since ODSI requires all devices to supply IP addresses that can be used as sources and destinations in bandwidth creation requests (see Signalling-address below).

2.2. ODSI Configuration Options

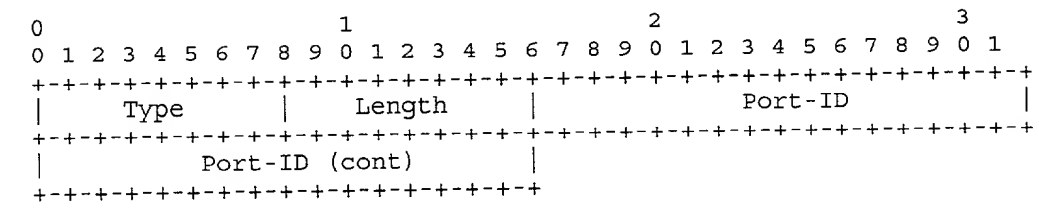
ODSICP Configuration Options allow negotiation of ODSI

parameters for the associated optical connection. ODSICP uses the same Configuration Option format defined for LCP [Ref2], with a separate set of Options.

Current ODSI configuration option values are assigned as follows. Vendor-specific options can be assigned using the procedures in [Ref6].

Type	Name	Required
1	Port-ID	yes
2	Signalling-address	yes
3	Sub-channel-size	no
4	Concatenation	no
5	Authentication-algorithm	no

2.2.1. Port-ID



Type

1

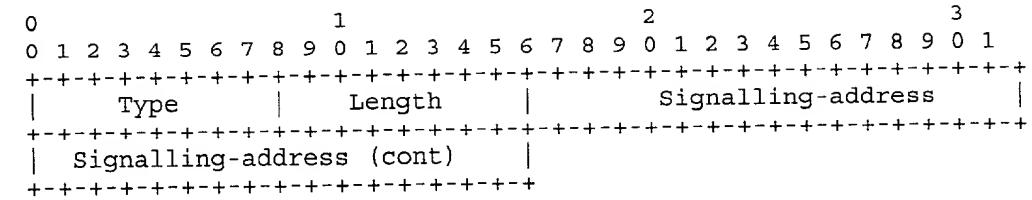
Length

4

Port-ID

The Port identifier to be used in bandwidth requests. This identifier uniquely identifies the port within the switch; and IfIndex can be used. When the port is capable of channelization, the Port-ID identifies the whole port, while a separate channel ID (not configured, but identified according to the port's physical layer standards; see for example Section A.1 of [Ref1]) identifies channels within the port. Port-ID is a required parameter.

2.2.2. Signalling-address



Type

2

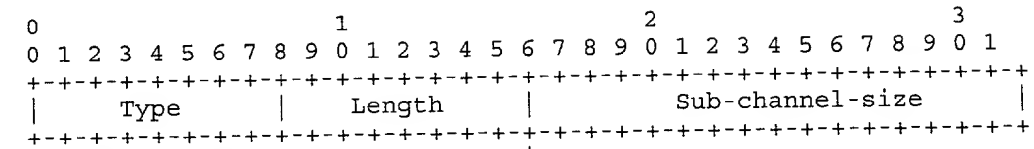
Length

4

Signalling-address

The IP address to be used as the destination of signalling requests (bandwidth creation, deletion, change and query) which concern the optical connection. Signalling-address is a required parameter.

2.2.3. Sub-channel-size



Type

3

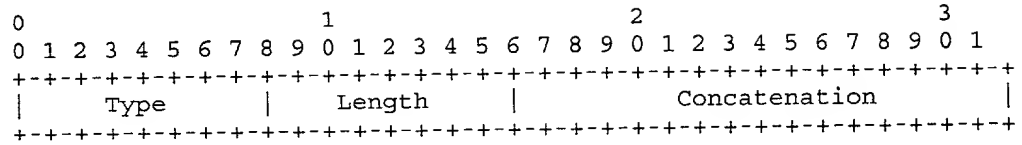
Length

4

Sub-channel-size

If the device is capable of channelization of the optical connection, this value specifies the minimum size of each channel. Size in given in IEEE 32-bit floating point. Example values of this field include OC-768 (0x511450c0), OC-192 (0x501450c0), OC-48 (0x4f1450c0), OC-12 (0x4e1450c0), OC-3 (0x4d1450c0) and STS-1 (0x4c45c100).

2.2.4. Concatenation



Type

4

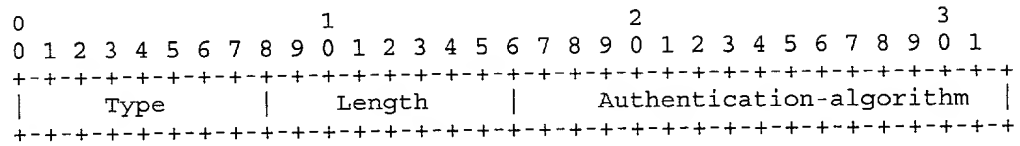
Length

2

Concatenation

When the optical connection is capable of channelization, this parameter indicates the ability of the device to concatenate channels together into a single logical channel. Values include 0 (no concatenation), 1 (standard concatenation) and 2 (arbitrary concatenation).

2.2.5. Authentication-algorithm



Type

5

Length

2

Authentication-algorithm

The authentication algorithm used to authenticate addresses registered with this port. Currently assigned algorithms include: 0 (none), 1 (MD5).

2.3. ODSI address registration

Endpoints are identified in bandwidth creation requests by IP

addresses. Addresses can be dynamically associated with an interface by the ODSICP message Register-Address. One or more previously registered addresses can be withdrawn from an interface using Withdraw-Address. Successful registration and withdrawals are answered with Register-Ack messages, while unsuccessful attempts elicit Register-Naks.

Every Register-Address message contains a list of addresses to associate to the port and the user group that the addresses belong to. Optionally, if authentication has been configured a message hash is also included. The same address can be registered on multiple ports, and in multiple user groups.

When the ODSICP first reaches Opened state, there are no registered addresses. At this point, addresses can be registered with Register-Address. Register-Address has the same format as Configure-Request, and much of the same semantics, except that:

(1) Register-Address can only be sent when ODSICP is in Opened state.

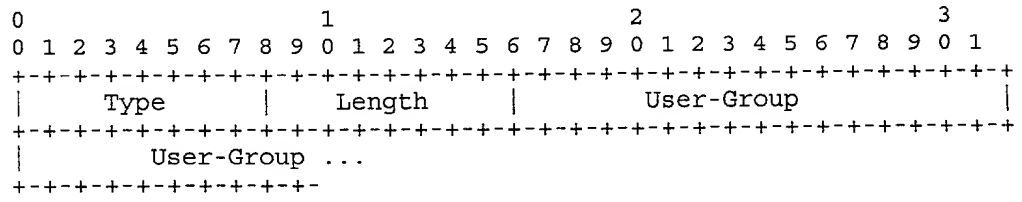
(2) Register-Address can be sent repeatedly, in which cases the results accumulate. For example, to register 100 addresses with a port, the internetworking device can send one Register-Address with 100 addresses or 100 Register-Addresses with one address each.

(3) The body of the Register-Address message holds options just like Configure-Request, only these options have separate type definitions as seen below.

The above considerations also hold for the Withdraw-Address message. The Options found within the body of the Register-Address and Withdraw-Address messages are as follows:

Type	Name	Required
128	User-Group	yes
129	Registered-Addresses	yes
130	Message hash	no

2.3.1. User-Group



Type

128

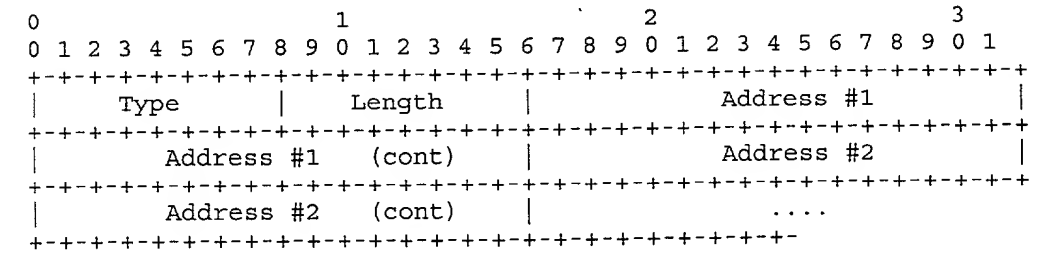
Length

variable

User-Group

An identifier for the community of devices to which the internetworking device belongs. For example, all the routers belong to an ISP would belong to the same user group. Bandwidth creation is restricted to devices belonging to the same user group. Address registrations and bandwidth creation is authentication and accounted for on a user-group basis.

2.3.2. Registered-addresses



Type

129

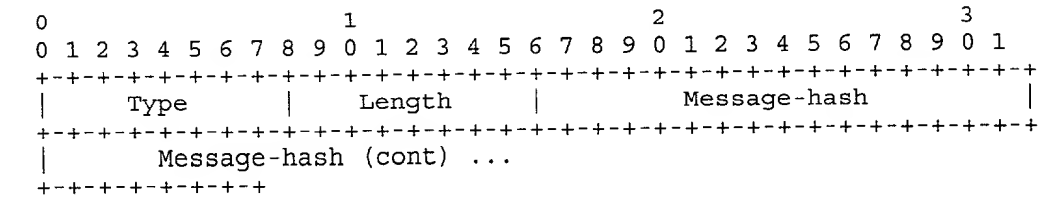
Length

multiple of 4

Addresses

A list of addresses which will identify the port in signalling bandwidth creation requests. Since the same address may refer to multiple ports, the Port-ID may have to be included to narrow the choice down to a specific port. In this case, if Port-ID is not included, the optical network will connect any matching port to an incoming bandwidth request.

2.3.3. Message-hash



Type

130

Length

variable

Message-hash

If authentication is configured for the port, all Register-Address and Withdraw-Address messages are verified by appending a one-way hash of the message contents and a secret key to the message.

References

[Ref1] ODSI Coalition, G. Bernstein, R. Coltun, J. Moy and A. Sodder, editors, "Optical Domain Service Interconnect (ODSI) Functional Specification", March 2000.

[Ref2] Simpson, W., "The Point-to-Point Protocol (PPP)", RFC 1661, July 1994.

[Ref3] McGregor, G., "The PPP Internet Protocol Control Protocol (IPCP)", RFC 1332, May 1992.

[Ref4] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[Ref5] Boyle, J., R. Cohen, D. Durham, S. Herzog, R. Rajan and A. Sastry, "The COPS (Common Open Policy Service) Protocol", work in progress, November 6, 1999.

[Ref6] Simpson, W., "PPP Vendor Extensions", RFC 2153, May 1997.

Editors' Addresses

Greg Bernstein
Ciena Corporation, Core Switching Division
10201 Bubb Road
Cupertino, CA 95014
Email: greg@ciena.com

Rob Coltun
Siara Systems
300 Ferguson Drive
Mountain View, CA 94043
Phone: (650) 390-9030
EMail: rcoltun@siara.com

John T. Moy
Sycamore Networks
10 Elizabeth Drive
Chelmsford, MA 01824
Phone: (978) 367-2161
Email: jmoy@sycamorenet.com

Arnold N. Sodder
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x110
Fax: 978-264-0671
Email: asodder@tenornetworks.com

K. Arvind
Tenor Networks, Inc.
100, Nagog Park, Acton, MA 01720
Phone: 978-264-4900 x110
Fax: 978-264-0671
Email: arvind@tenornetworks.com

[Page 16]

APPENDIX V

ODSI Coalition
ODSI Signaling Control

Version 1.4.5

K. Arvind
A. Sodder
Tenor Networks

Y. Cao
J. Moy
Sycamore Networks

R. Coltun
Redback Networks

H. Epstein
Y. Xu
Lucent Technologies

R. Hartani
Caspian Networks

B. Rajagopalan
Tellium, Inc.

G. Grammel
W. Zimmermann
Alcatel

K. Mehadji
F. Tillerot
France Telecom

Editors

Optical Domain Service Interconnect (ODSI) Signaling Control Specification

Table Of Contents

1.0 Introduction

The Optical Domain Service Interconnect (ODSI) signaling control protocol defines an interface to the optical network, whereby devices, such as routers, SONET/SDH add/drop multiplexers (ADM), and ATM switches can dynamically request bandwidth from the optical network [Ref1]. ODSI thus essentially provides a User to Network Interface to the optical network. More specifically, the ODSI signaling control protocol is used to request the establishment of an optical "trail" between two trail endpoints, delete an existing trail, query the status of a trail or modify the resources associated with an existing trail. It provides an interface, via a single control point, to a signaling mechanism, which is used within the optical network. Note that this document uses the term "trail" to denote the end-to-end connection between two devices attached to an optical network (often called an optical circuit).

ODSI signaling control is designed to allow optical edge device to participate in trail establishment; it is optimized for third-party signaling control. The protocol is designed to work both in-band and out-of-band. This document specifies the procedures for, and interactions between the control points which constitute the ODSI signaling control. The signaling control functions are referred to as transactions.

The ODSI model may be categorized as belonging to the Domain Services Model and the Overlay Interconnection Model described in [Ref2]. In this respect, the ODSI signaling interface is similar to a traditional user to network signaling interface. The main function of the user to network signaling interface in this model is to transport service requests and responses between a user and network domain, which are assumed to be otherwise disjoint from a control plane perspective. The ODSI signaling protocol provides this transport function.

While the information transported across the user to network interface is similar to a traditional UNI [Ref3], there are differences in the actual mechanisms (the signaling protocol) used to transport the information. ODSI signaling control uses a TCP application that is specifically designed for transporting signaling information between signaling entities. This protocol is the subject of this document. In addition to providing a reliable signaling transport, the ODSI signaling control protocol also addresses the following additional requirements unique to ODSI [Ref1]:

- o ODSI clients may be IP routers, ATM switches, SONET/SDH devices, and in general any client of the optical network. IP support in many of these client devices, may be restricted to a light-weight IP stack. A specialized TCP application such as the ODSI signaling control protocol would provide a low-burden approach to transporting signaling requests for these devices.

- o ODSI supports third party signaling control, where a third party device such as a network management station may set up connections between user devices via ODSI signaling control. Third party signaling control introduces special ordering requirements that the ODSI signaling control model addresses.

ODSI signaling control also provides other advantages such as extensibility (the ordering of signaling entities involved in a transaction can be varied), flexibility (user devices involved in a transaction can coordinate with each other before sending a

request to the network), and a single point of interaction with the network. This transaction model frees the optical network to implement its circuit setup in various ways (distributed, standard NNI, proprietary NNI, centralized, etc.), allowing the optical network to evolve independently from the signaling protocol.

2.0 Overview

There are four control points that participate in ODSI signaling control.

- Trail Control Requester
  The Trail Control Requester originates ODSI transaction requests and receives an indication that the transaction has been successfully completed or failed.
  The Trail Control Requester could be any user device, which does not need to be a trail endpoint, that is connected to the optical network such as a management station, or SONET/SDH ADM, or a router or a switch which is a trail endpoint.

- Trail Control Head
  The control point associated with one end of the optical trail. The Trail Control Head denotes the first trail endpoint device to participate in the transaction procedures. Note that different transactions performed on the same trail (e.g., a trail create followed by a trail modify) may choose different ordering of control points for a particular transaction.

- Trail Control Tail
  The control point associated with the other end of the optical trail. The Trail Control Tail denotes the last trail endpoint device to participate in the transaction procedures. Note that different transactions performed on the same trail (e.g., a trail create followed by a trail modify) may choose different ordering of control points for a particular transaction.

- Optical Network Controller (ONC)
  The optical network is composed of a set of interconnected optical switches with the ability to create bi-directional optical trails. The ONC is the control point for a physical or logical entity or a set of entities that participates in ODSI signaling on behalf of the optical network.

  It is envisioned that the ONC verifies the policy for trail action. Note that the ONC may itself implement the verification policies or may request verification [Ref6].

ODSI signaling control is intrinsically third-party; neither of the trail endpoints (the Trail Control Head or the Trail Control Tail) need be the device that initiates a transaction. However, a device may serve as Trail Control Requester as well as Trail Control Head.

ODSI transactions do not carry (and is not identified by) an MPLS-like Forwarding Equivalence Class (FEC) [Ref4] in the request messages, nor does it designate how the user device should relay the data outside of the optical network. Instead,

the trail created by an ODSI transaction becomes the optical "mid-section" of a circuit (or Label Switched Path (LSP)) with the devices associated with the Trail Control Head and Trail Control Tail being the boundaries of the mid-section. These devices may constitute termination points of the optical mid-section (e.g., the Trail Control Head and Trail Control Tail are routers), or may be devices which are part of a longer circuit that is traversing the optical network. After the trail setup has been successfully completed, the Trail Control Head and Trail Control Tail bind the appropriate data channels (e.g., circuit, IP destination, etc.,) to the newly established trail. The trail is identified by the Trail ID.

ODSI signaling control messages are IP messages. ODSI messages can be forwarded on an IP data path between the requesting device (e.g., the Trail Control Requester) and one of the optical switches acting as the ONC. IP forwarding may be through an in-band control channel, or it could be through out-of-band connectivity provisioned for control traffic. For example, this could be a 10-base T Ethernet connecting the user device and the optical switch. ODSI signaling control messages may not follow the same path as the trail that will be established.

The ODSI signaling control functions are referred to as transactions; each ODSI transaction results in a temporary transaction chain that spans the ODSI devices that participate in the transaction (see section 6.0 ODSI Transaction Chains for more details). The devices that participate in the transaction, the order of participation and the control function that each take on (e.g., Trail Control Head or Trail Control Tail) is defined in the Trail Control Points field of the transaction messages. The number of devices participating in the transaction will differ according to the transaction type and desired mode (see section 6.3).

3.0 ODSI Signaling Transactions

The ODSI signaling control protocol specifies the following four types of signaling transactions.

- o Create
  Create a point-to-point optical trail between two user devices connected to the optical network. The two endpoints of the trail and its characteristics (i.e., resource parameters) must be specified. The trail is identified by a Trail ID.

- o Modify
  Change the characteristics of an active trail. The trail is identified by a Trail ID. For example, a modify might be used to increase or decrease the amount of bandwidth associated with an existing trail.

- o Query
  Query the status of an existing trail. The trail is identified by a Trail ID. The ONC or one of the endpoints of the trail is the target of this operation. The status (established or not) and the endpoints of the trail are returned as the result of this operation.

- o Delete
  Delete an existing trail. The trail is identified by a Trail ID.

Each of the four ODSI control points play a unique role in the management of optical trails. Note that since the underlying transport for signaling is reliable (ODSI uses TCP as the inter-device transport mechanism), message acknowledgments are used to communicate that a transaction has been completed, and a TCP connection is still active but not that a message has been received (which is handled at the TCP layer).

An example of the create transactions is as follows. As a result of a network management action a CREATE message is originated by the Trail Control Requester and sent to the Trail Control Head which processes the request and sends it to the Trail Control Tail. The Trail Control Tail processes the request and send it to the ONC. The Trail Control Head and Trail Control Tail fill in information that is specific to the associated user device.

When the ONC receives the request, it validates the specifics of the request and attempts to allocate an optical trail with the requested characteristics. If such a trail cannot be created, the ONC sends a NOTIFY message to the Trail Control Requester, in the reverse direction along the control path, to inform the control points that the request has not succeeded. Otherwise, the ONC sends a CREATE ACK message back along the control path to inform the control points that the request has succeeded. It is envisioned that the ONC verifies the policy for trail action [Ref6] [Ref14].

The trail is usable by the Trail Control Head and the Trail Control Tail when they receive a "circuit up" event, which may arrive before the acknowledgment has been received.

3.1 Modes Of Operation

ODSI signaling does not restrict the control of optical network signaling to a single control model. In doing so, it facilitates independent evolution of the control functions for devices inside and outside of the optical domain. This section discusses some of the modes that ODSI signaling control may run in. Appendix B gives some examples of common modes.

3.1.1 Control Points

The Trail Control Head and Trail Control Tail may be the control points for various types of devices. For instance, they may be routers, layer 2 devices (such as a device which supports ATM, Frame Relay or Ethernet switches) or MPLS switches. The head and tail may be layer 1 devices (SONET/SDH ADMs or an optical cross-connect) that are feeding into a higher-capacity optical core. Alternatively the head and tail may be devices that are inter-connecting a metro-area network and a long-haul network.

The control point for the ONC may be a (or a set of) management station which allocates optical trails on behalf of the optical network, or may be an optical switch (e.g., the one connected to the Trail Control Head or Trail Control Tail) which is using MPLS for control of optical networks. In the latter case, it's always the optical edge switch connected to the Trail control head that makes the ONC decisions.

The Control Points field must always include a requester (the first control point in the Control Points Field). The Trail

Control Requester may be the same device as the Trail Control Head (i.e,. no third-party requester). In this case there will be two entries in the Control Points fields with the same IP address but differing control functions; one representing the Trail Control Requester and one representing the Trail Control Head.

Each control point will process the action and relay the message to the next control point as listed in the Control Points field of the transaction. A device may support different control functions (e.g., for different user groups). For example, a device in a metro network may be the Trail Control Tail for trails allocated within the metro network but may function as both the Trail Control Requester and Trail Control Head when a trails is allocated from a metro network traversing a long-haul network. Note that the mechanisms (e.g., single or multiple transaction chains) for metro and long-haul interaction have yet to be defined.

The control point functions are fairly simple. The control point must verify that it can assume the control function in the received message (as designated in the Control Points field), process the request and, if it does not assume the role of a terminating device, relay the message to the next device in the control points field.

3.1.2 Control Points path order

Somewhat analogous to an explicitly-routed control setup used with MPLS and PNNI (where the initiator defines a the path that signaling will use), the Trail Control Requester (albeit a much less complex function than that of the initiator of an MPLS explicitly-routed LSP), defines which control points will be used and the order of the control points. The following describes three different control points path orderings.

ONC as the last control point
[requester->head->tail->ONC]

One quality of this ordering is that it allows verification and coordination of edge-device information (e.g., channels, ports, user groups) before sending the request to the ONC.
See Appendix B.2 and B3 below for examples of this ordering.

Note that with an additional object or two, this model can be used to build inner tunnels using optical trails as trunks.
For example if a service is offered to provide Ethernet VLAN services over an optical network ODSI signaling can be used to assign and coordinate VLAN tag information between the edges, specifying the optical trail to be used as the VLAN's trunk.
The trunk may be shared by multiple VLANs.

Third-Party Initiated Traditional UNI
[requester->head->ONC->tail]

This ordering most resembles the traditional UNI with a third-party requester. Under this ordering, the Trail Control Requester requests a trail.
The Trail Control Head relays the requests to the ONC. The ONC informs the Trail Control

Tail after creating a new trail. See Appendix B.1
below for an example of this ordering.

Requester and ONC
[requester->ONC]

This ordering assumes that devices at the edge of
the optical network are manually configured
to use specific ports and channels, while
the optical trail is allocated or modified
by the requester communicating directly with
the ONC.

3.1.3 Control Network

Since ODSI signaling control uses TCP as a transport between control devices,
it requires an IP network for transport of control
message between devices. The IP control network
may be embedded in the optical network (i.e., optical switches
have implemented a low capacity IP routing function)
or may be a completely distinct control network.

The communication path between a user-side control device and
the control network may be an in-band
or out-of-band connection. An in-band connection may be, for
example, realized using SONET/SDH overhead bytes.
An out-of-band communication path may be, for example
a 10-base-T Ethernet that is dedicated for relaying control
information between the user-side and the optical network control points.
Edge devices must have sufficient knowledge of the control network
so as to be able to route to peer control devices
and to process transactions.

4.0 Message Types

The following message types are used in ODSI signaling transactions:

- CREATE

  This message is originated by the Trail Control Requester,
  and relayed along the Trail Control Points from one
  control device to the next after each device has completed
  processing the message. It includes enough information for
  the ONC to build an optical circuit within the optical
  network. The Trail Control Points that may be involved
  in the creation of a trail are the Trail Control Requester,
  Trail Control Head, Trail Control Tail and ONC.

  In response to receiving the CREATE message the Trail
  Control Head assigns a trail ID and informs the Trail
  Control Requester of the Trail ID by sending it a NOTIFY
  message (see below). The contents of the CREATE message is
  modified as the message progresses along the
  Trail Control Points. Specifically, the Trail ID is
  filled in by the Trail Control Head and termination
  port information may be filled in by the Trail Control
  Head and Trail Control Tail.

  Any device might append or modify the vendor-specific
  data field, which is required to be the last field in
  the message.

- CREATE ACK
  This message is originated by the ONC in response to a successfully completed CREATE message. The ACK message is relayed in the reverse order of the CREATE message. The CREATE ACK message is identified by the upstream node via the Trail ID.

- MODIFY
  This message is originated by a Trail Control Requester, and relayed along the Trail Control Points, from one control device to the next after each device has completed processing the message. It includes enough information to modify the trail resources associated with an existing optical trail within the optical network.

- MODIFY ACK
  This message is originated by the ONC in response to a successfully completed modify request. It is relayed in reverse order from the MODIFY message. The MODIFY ACK message is identified by the upstream node via the Trail ID.

- QUERY
  This message is used to obtain the status of an existing trail. The trail is identified by its Trail ID. The ONC or either one of the endpoints (Trail Control Head or Trail Control Tail) is specified in the Control Points field. The status (established or not) and endpoints of the trail are returned in the QUERY RESPONSE message.

- QUERY RESPONSE
  This message is originated by the ONC, Trail Control Head or Trail Control Tail in response to a QUERY message. The status (established or not) and endpoints of the trail are returned. The QUERY RESPONSE message is identified by the upstream node via the Message ID TLV (containing the message ID generated by the upstream node for the corresponding QUERY)

- DELETE
  This message is originated by a Trail Control Requester, to delete an existing trail. The trail is identified by a Trail ID. The Control Points for a delete may consist of the Trail Control Requester, Trail Control Head, Trail Control Tail and ONC. Either one of the endpoints may be the first device in the Trail Control Points field.

- DELETE ACK
  This message is originated by the ONC in response to a successfully completed DELETE message. It may include vendor-specific information. The DELETE ACK is relayed in the reverse order of the DELETE message. The DELETE ACK message is identified by the upstream node via the Trail ID.

- KEEPALIVE
  After a TCP connection is established, the initiator and peer exchange KEEPALIVE messages. A KEEPALIVE message includes a PeerDead time, which is the time period that the peer device shall wait to receive a KEEPALIVE or transaction message before transitioning the TCP connection to the down state.

KEEPALIVEs are mandatory to send at the beginning and the end of a TCP sessions only.(see section 6.4)

- o NOTIFY
  A NOTIFY message is used for several purposes:

  1) Originated by the Trail Control Head (or ONC if there is no head in the transaction's control points) in response to receiving a Trail CREATE message. It serves to inform the Trail Control Requester of the Trail ID for subsequent transactions.

  2) As a response to an illegal value in a KEEPALIVE or transaction message.

  3) In response to Create and Modify transactions when the allocation for the request cannot be completed.

  4) In response to the failure of a trail within the optical network, the ONC will send a NOTIFY to its predecessor.

  The NOTIFY message is identified by the upstream node via the information included in the Notification Association TLV (see section 6.5.5).

  See section 6.5 for details of the notification procedures.

The contents of these messages are described below, followed by the description of device-specific procedures.

5.0 Message Contents

The following describe the information fields contained in the ODSI transaction messages. Appendix A defines the packet formats, parameter values and encoding. Notification procedures and NOTIFY message contents are defined in section 6.5.

5.1 Message and Transaction Control

- o Message Type
  Identifies the message type.

- o Message Length
  Specifies the cumulative length in octets of the message including the Message ID, Mandatory Parameters, and Optional Parameters.

- o Message ID
  32-bit value used to identify this message. The Message ID is used by the sender to facilitate identifying notification messages that may apply to this message. A device sending a notification message in response to this message shall include this Message Id in the notification message; see section 6.5 for more information on the notification procedures.

The Message ID has per-peer significance. A device allocates a message ID for each message it relays to its peer.

o Trail Control Points TLV

This field specifies the ordered list of control points that the message will sequentially travel during the processing of the transaction. Each device in the list is identified by the control function (e.g., the ONC) and its IP address.

The order and number of control point devices is set by the Trail Control Requester. A Trail Control Requester must always be the first device present in the field. When a device takes on a dual control function (e.g., the Trail Control Request and Trail Control Head), both functions are present in the list; the IP address is listed twice with two control functions.

Note that the notion of head and tail are somewhat ambiguous in a bi-directional trail. For example, a create might choose the device list to be devices R, H, T, O (or R, H, O, T), whereas a subsequent trail modify of the same trail might choose the list to be devices R, T, H, O (or R, T, 0, H).

For a trail query transaction the list consists of two entries; the Trail Control Requester and one of the ONC, Trail Control Head or the Trail Control Tail.

o Trail Control Index

The Trail Control Index field is an integer which indicates which of the control entities is to be processing the message next. It is used by the device receiving the request

- to determine which control function it must assume for transaction (e.g., the Trail Control Head or Trail Control Tail)

- to determine the IP address of the predecessor and successor devices and to determine if it is the terminating device for the transaction.

When the message is relayed, the index field is incremented as the transaction progresses and is decremented as the transaction regresses (as with the Network Related NOTIFY messages).

o Integrity TLV

Corrupted or spoofed control requests could lead to theft of service by unauthorized parties or to denial of service caused by locking up network resources. ODSI protects against such attacks with a hop-by-hop authentication mechanism using an encrypted hash function. The mechanism is supported by Integrity objects [Ref11] that appears in all ODSI messages. These objects use a keyed cryptographic digest technique.

As in RSVP, the Integrity TLV is optional in a ODSI message, and

a ODSI control point may choose to ignore an included Integrity TLV. However, a ODSI control point receiving a message without an Integrity TLV is free to reject the message for this reason. This flexibility ensures that there is a means for ensuring security, but allows security to be enforced only where it is felt necessary. For example, ODSI peers within a common trust domain may choose to not include the Integrity TLV in ODSI messages.

More secure authentication mechanisms will be introduced in the future releases.

5.2 Endpoint Identification

- o Trail Head End Address
- o Trail Tail End Address

  The IP v4 address of the trail's endpoints. It is represented as a 32-bit value in network-byte order. Note that the IP v4 address of the trail endpoint in a device need not be the same as the address which is used to represent a data termination point.

- o Trail Head End Port
- o Trail Tail End Port

  The device's port and channel information. This TLV consists of a port index, and if needed the channel identification. The port index identifies a particular physical interface connecting into the Optical core. In IP MIBs, this is commonly an IfIndex.

  The channel is used when establishing sub-rate bandwidth connections on multiplexed interfaces. It identifies which sub-rate channel should be connected. For example, on a SONET/SDH interface this would be a time slot, as specified in [Ref5], [Ref13].

  When used in a CREATE message, this field may be filled in by the Trail Control Head to communicate it's desired port information to the ONC. If the field is not present, the optical network will choose a port and channel associated with the Trail Control Head's IP address. If the ordering requester->head->ONC->tail is used, Trail Tail End Port field must be filled before the CREATE message is received by the ONC.

- o Trail User Group

  Identifies the user group for the trail. This is a further qualification of the Trail Control Head and Trail Control Tail's IP address. A trail can be created only if the head and tail are associated with the same user group. The notion of a user group is similar to the closed user group concept in X.25. For example, if there were 10 ISPs whose routers were all attached to the same optical network, each ISP would have a separate user group identifier.

5.3 Trail Specific Information

- o Trail State

This value is present in the QUERY RESPONSE message and denotes the state of the trail which may be Trail Active, Trail Pending (waiting to be brought up), Trail Inactive or Trail Unknown (there is no record of this trail). The trail state is present in the Qualifier field of the Trail ID TLV.

o Trail ID
This is a unique identifier of a ODSI trail within the network. The Trail ID is composed of a device's own IPv4 addresses and a 16-bit ID which is locally unique to the device. The Trail ID is used by all transaction types to identify the trail. The Trail ID is used by the Trail Control Head and Trail Control Tail to bind the input and output forwarding actions (e.g., perform IP forwarding) to and from the trail.

In a trail create transaction, the Trail ID field is not present in the message as originated by the Trail Control Requester when it is transmitted to the first device in the Control Points field because the Trail ID as assigned by first control device.

When a CREATE message received and the device is the Trail Control Head or there is no Trail ID in the message and the device is the ONC, the device allocates a Trail ID and sends a NOTIFY message back to the Trail Control Requester indicating a Trail ID. The Trail Control Head inserts the Trail ID in the CREATE message before the message is relayed to the next device in the Control Points field.

The Trail ID field is set by the Trail Control Requester in all other transactions.

5.4 Bandwidth Characteristics

The characteristics of the dynamic bandwidth channel connecting two user devices can be specified by the following parameters.

o Service Level
Service level encompasses priority (whether the circuit can be preempted by other, higher priority circuits), protection (whether the bandwidth should be protected against failures, and if so, the speed at which the protection will restore bandwidth service after a failure), and availability. Service level is encoded as an integer, whose meaning is assigned by the optical network provider. For example, one provider could encode its "gold service" as service level 1, which might be advertised by the provider as "50 millisecond restoration through BLSR technology".

o Framing
This parameter indicates the format of the signal on the port between user and network devices. It is defined to be one of:

- PDH
- SONET

- SDH
- Digital wrapper
- LAN ethernet
- WAN ethernet. This is transported in an OC-192c.

The framing may be left unspecified by a CREATE transaction when implied by endpoints of the trail. The CREATE ACK will include the resulting framing. If supported by the network, framing at the two service endpoints may be different.

Note that the same amount of bandwidth may be framed in different ways (i.e., bandwidth does not imply a framing format).

This field is not used with trail modify transactions as the framing may not be changed on the fly.

o Transparency

This parameter is interpreted with respect to the framing. It indicates the portion of the signal that is available for user data traffic. For SONET/SDH framing, the transparency options are:

- PLR-C. Physical layer regeneration, meaning all the SONET/SDH signal is available for user data.
- STE-C. Endpoints are attached to section-terminating equipment. Only line and path overhead are available to user data traffic.
- LTE-C. Endpoints are attached to line-terminating equipment. Only path overhead is available to user data traffic.

o Bandwidth Size

The size of an optical trail as defined in bits per second. Any size bandwidth may be requested, although it is not required that all sizes be supported by the optical network. When a size is requested that the optical network does not support, it may return a larger bandwidth connection, but never a smaller. However, if the amount of bandwidth requested is not available, the optical network will indicate the maximum amount of bandwidth that can be obtained to the specified endpoint in a NOTIFY message.

Examples include STS-x, STM-x, OC-x, VC-x, 1 or 10 megabit (Ethernet). Bandwidth size may be left unspecified when implied by endpoints of the bandwidth channel.

o Diversity

A CREATE transaction may request that the new circuit be diverse (that is, share no common facilities) from an existing set of ODSI-created trails.

The encoding of this characteristic is defined as a list of one or more existing trail, specifying for each trail:

- The Trail ID
- The kind of diversity required from this existing

circuit, one of (a) link diverse, (b) node diverse, or (c) SRLG (Shared Risk Link Group as defined in [Ref10]) diversity .

The diversity type is noted in the Trail ID TLV, which is nested in the Diversity TLV.

- o Directionality
  This bandwidth characteristic allows for both unidirectional and bidirectional trail. In addition, it allows the specification of asymmetric trails, by specifying such trails as two unidirectional halves.

- o Contract ID
  This parameter provides billing and authorization information for the trail. Contract ID is encoded as a string, and is assigned by the provider of the optical network. It is included in bandwidth requests, and in that way is passed to the optical network so that it can verify whether the requester is allowed to set up a particular trail (e.g., does the requester have enough credit in their account).

- o Vendor Specific Information
  An optional opaque field used to pass vendor-specific information. This may be used by vendors to create proprietary bandwidth descriptions.

6.0 ODSI Transaction Chains

Each ODSI signaling transaction builds a transaction chain which spans the devices (i.e., control points) that participate in the transaction as specified in the Control Points field of the transaction's messages. The transaction chain as a whole, functions as a multi-device client-server transaction. An ODSI transaction chain is short lived, remaining active only for the life of the transaction. The trail create, trail modify and trail delete transactions may span several devices, whereas a trail query transaction only includes the Trail Control Requester and one other device. The transaction chain is built incrementally as the transaction progresses from the first to the last specified control point. When the transaction has been completed (or a failure has occurred), the transaction chain is disassembled. A record of the trail state is maintained by the Trail Control Head Trail Control Tail and ONC for the life of the trail. The record is identified by the Trail ID.

Each device that is part of a transaction chain has a client-server relationship with the devices that it peers with. When a device receives a transaction message from its predecessor, it processes the message and relays the transaction message to its successor device, opening a transport connection to its successor device if necessary. The relayed message constitutes a requests for an action (e.g., CREATE, MODIFY, DELETE, QUERY) to it's peer device. When the device receives a reply it completes its part of the transaction and relays the reply to its predecessor.

ODSI transactions require reliable and in-order delivery of messages. To satisfy these requirements, ODSI transaction chain uses TCP as the inter-device transport mechanism for messages that are relayed between devices. Multiple transactions can share an active TCP connection. ODSI devices use TCP port 1308 for establishing its connections. Note that it is common practice

for devices (see [Ref7]) which use TCP as a transport to use a loop-back IP interface to receive TCP connection request as this make the device more resilient to the failure of any single interface.

Only a single modify-class transaction (create, delete or modify) may be performed at one time on a single trail. For example, a trail delete may not be invoked until a previously invoked trail modify has been completed.

Transaction state is maintained at each device along the transaction chain for the life of the transaction. When a new event is received, a transaction control block (TCB) is created. The TCB is bound to the TCP connections (one to the predecessor device and one to the successor device) and stores the state of the transaction. Section 6.2 below gives a more complete description of the TCB. Note that the TCB structure definition is presented to describe general functionality and does not necessarily represent an implementation requirement.

6.1 Procedures for ODSI Control Points

The devices that participate in ODSI (Trail Control Requester, Trail Control Head, Trail Control Tail and ONC) are referred to as control points as they specify the singling control points for an optical trail. The set of control points for a transaction, as well as the order of their participation, is specified in the Control Points field of the transaction.

The function of the ODSI process is to manage inter-device transport (using TCP), wait for incoming events, initialize transactions when events inform the process to do so, process transaction events for active transactions and terminate transactions as they complete.

The procedures and state transactions differ for each of the participating devices. The following sections describe the functions performed by the ODSI Control Points.

6.1.1 Trail Control Requester

Upon initialization, the Trail Control Requester waits on events to request the initialization of a transaction. The requester formats a request message filling in the fields that have been specified by the management request.The Trail Control Points field in the message specifies the devices that participate in the transaction. Each device is identified by an IP address.

If the transaction is a modify-class transaction (trail modify or trail delete), the device checks to see if there is modify-class transaction pending for the specified trail (using the Trail ID). Note that procedures for a Trail Create transaction includes the allocation of a new Trail ID; a device cannot check for a modify-class collision when the transaction is a Trail Create. If there is a modify-class collision, the device does not continue processing the event and should log the event with a notification.

If the event is a create, the requester will be notified of the Trail ID from the successor device (i.e., the first device listed in the Control Points field), via a NOTIFY message immediately after the successor device receives the CREATE message. The

Trail Control Index is set to 2 to indicating the next device to process the message. The device then enters TransactionConnect state, performing specified procedures.

When the message is relayed to its successor device, the requester initializes the TransactionDead timer and waits for a response to the transaction. The exact value of the TransactionDead timer is a local matter, but should be sufficiently large to allow the transaction to complete. If the timer expires, the transaction send a query message to the ONC (and perhaps head and tail) to see what state the transaction appears to be in; the device should log the event with a management notification. The device may originate a DELETE message to clean up the transaction state as well as free any connection resources tied to the failed transaction.

6.1.2 Transit and Terminating Devices

Upon initialization, the ODSI process begins waiting on TCP port 1308 for incoming TCP connections and incoming messages (see section 6.4 ODSI Inter-Device Transport below).

Upon receiving a message, the device (if it is the trail head or trail tail) verifies that it is part of the User Group specified in the message. If the device is not part of the specified user group, the device follows the notification procedures specifying the error code "Unknown User Group". Note that a user group is associated with specific ports on the device; the user group specifies which ports are to be used for the transaction ([Ref12]).

If the Integrity TLV is present, The device authenticates the sender (predecessor) of the message, verifying that the message has not been originated by an unwanted (accidentially used) device by verifying the Integrity field. If the authentication fails then the device follows the notification procedures specifying the error code "Authentication Failure".

The device should verify that it can correctly perform the control function that the Control Points field specifies for the device (e.g., the Control Points field may incorrectly denote a router as being the ONC). Note that a device's control function may differ depending on the User Group or other criteria.

When a device receives the first message of a new ODSI transaction, it initializes an ODSI transaction control block (TCB), and transitions to the state TransactionConnect. If the first message of a transaction identifies the transaction as being a modify or a delete, the device checks to see if there is modify-class transaction (create, modify, delete) pending for that Trail ID. If there is, the device follows the notification procedures specifying the error code, "Modify Detected".

If the received message type does not correspond with the expected message type (e.g., the wrong response type has been received) the device follows the notification procedures specifying the error code, "Incorrect Message Type".

The device then continues to parse the message according to the current state and contents of the message. If the contents are invalid (contains an illegal value, cannot be parsed or if particular fields are missing), the device follows the notification procedures specifying the error code, "Parse Error"

or "Illegal Value".

6.2 Transaction Control Block (TCB)

ODSI Signaling uses a minimal state machine to maintain transaction states. The state machine is used in conjunction with the Procedures for ODSI Control Points (section 6.1), Notification Procedures (section 6.5), the Transaction Control Block (TCB) and section entitled ODSI Inter-Device Transport (section 6.4).

Transaction state is maintained at each device along the transaction chain for the life of the transaction. At each device along the chain, a transaction control block is created. The TCB is bound to the predecessor's and successor's TCP connections. As messages and TCP events are received the actions are taken according to the transaction state.

The TCB can be identified by a combination of the Trail ID and the transaction type. The TCB should also be able to be referenced by the Message ID. Note that this data structure definition is presented to describe general functionality and does not necessarily represent an implementation requirement.

- o Trail ID
  The identifier for the trail and (along with the Transaction Type) the index used to access the TCB.

- o Transaction Type
  This item identifies the type of transaction. The transaction type could be one of the following:

  - TrailCreate
    The transaction is a request to create a new trail between two user devices connected to the optical network.

  - TrailDelete
    The transaction is a request to delete an existing trail. The trail is identified by a Trail ID.

  - TrailQuery
    The transaction queries the status of an existing trail. The trail is identified by a Trail ID. Either one of the trail endpoints can be specified. The current status (established or not) and other endpoint of the trail is returned.

  - TrailModify
    The transaction is a request to increase or decrease the amount of bandwidth associated with active trail. The trail is identified by a Trail ID.

- o Message IDs
  32-bit value used to identify the a message. The TCB stores up to two Message IDS. One for messages received by a peer and one for messages that are sent to a peer. Note that the Message ID has per-peer significance. That is, the device's predecessor allocates the Message ID for the message that it relays to the device.

The message ID is used by a device originating a NOTIFY message to identify a trail transaction event or message error to a peer. See section 6.5 for more information on the notification procedures.

- o Transaction State
  This item identifies the state of the current transaction. The type of transaction is identified by the Transaction Type. See section 6.3 for a list of the transaction states.

- o Trail Control Points
  The ordered list (and number) of devices specifying the Control Points that the message will travel when visiting all the devices involved in the processing of the transaction. Each device in the list is identified by a Control Function (what role the device must perform) and its IP address.

- o My Index
  The index to the device's entry in the Trail Control Points field. Along with the Trail Control Points field, this index indicates the devices control function in the transaction (e.g., Trail Control Head, Trail Control Tail, etc.).

- o Inbound TCP connection ID
  This entry references the TCP table entry's TCP connection to a peer as initiated by the peer device. The peer device is a predecessor in a transaction. The TCP connection will be terminated by the predecessor.

- o Outbound TCP connection ID
  This entry references the TCP table entry's TCP connection to a peer as initiated by this device. The peer device is a successor in a transaction. The TCP connection will be terminated by the predecessor.

- o TransactionDead timer
  The timer used to stop waiting for a transaction to succeed. If the timer expires, the transaction state is cleared.

- o Requested Attributes
  The set of path attributes requested by the transaction including items like bandwidth size, propagation delay trail priority and trail protection.

- o User Group
  The user group of this transaction. Note that the device may belong to more than one user group and that the transaction is specific to one of the user groups.

6.3 ODSI Transaction State Machine

6.3.1 TransactionConnect

The device enters this state when a new transaction event occurs (i.e., it has received the first message of a transaction or it is the Trail Control Requester and a management event has initiated a transaction).
Upon entering this state, a TCB is allocated, initialized and the predecessor TCP connection ID in the

TCB is associated with the inbound TCP connection.
If the device is the Trail Control Requester
and the transaction has been initiated by a management event,
there is no TCP association for the TCB's inbound TCP connection.

As with the reception of any transaction message,
the PeerDead timer is reset to its
time out value when a transaction (or KEEPALIVE)
message is received.

If the message is a CREATE message
and the device is the Trail Control Head or
there is no Trail ID in the message and the device
is the ONC, the device allocates a Trail ID and sends
a NOTIFY message back to the Trail Control Requester
indicating a Trail ID.
This is so the Trail Control Requester can identify the
yet-to-be-formed trail using the Trail ID
(e.g., in the event that the transaction does not complete
and the requester wishes to perform a query on the trail).

All devices store the Trail ID, Transaction Type and the
Trail Control Points
field in the TCB, noting the device's control function for the
transaction (e.g., Trail Control Head, Trail Control Tail, etc.).

If the device is a terminating device for
the specified transaction type, the device
begins processing the message.
The device will be a terminating device
if it is the last device listed in the
Trail Control Points field. For example, the device
is the Trail Control Head or Trail Control Tail and it has
received a QUERY message.
While processing the event, the device
enters the state TransactionProgressing.
Note that if the transaction can be processed immediately
(e.g., a QUERY) the device may respond immediately
to the transaction and progress directly to TransactionComplete.

If the device is not a terminating device for
the transaction, when entering the state TransactionConnect
if a TCP connection is open to the successor device (i.e., the next
control point indexed in the Control Points field)
the device relays the message and transitions to
the state TransactionProgressing.
Note that a transit devices may add information
(e.g., head or tail port information or vendor specific information)
to the message before relaying the message to the successor device.

If there is no TCP connection open to the successor device,
the device will initiate a TCP connection to its successor
(see section 6.4.4 Initiating TCP connections below).
The TCB's successor TCP connection ID is associated with
the newly-established outbound TCP connection.
While the TCP connection is being established, the
message is queued on the TCB's outbound queue
pending the acceptance of the connection to the peer.

When the TCP connection becomes operational,
the TCP connection procedure will notify the TCB
to relay the message to the successor device; the TCB
remains in the current state until the message is
relayed to the successor device.

The connection becomes operational after a device has
sent a KEEPALIVE to its peer and has received (and accepted)
a KEEPALIVE messages from its peer.
The reception and acceptance of the KEEPALIVE shows that
the connection is viable and since the KEEPALIVE contains the
authentication for the peer, it verifies that the peer is valid.
When the connection becomes operational, the
state is progressed to TransactionProgressing.

If the transaction message requires the device to wait
for a response, (i.e., the device is not a terminating device
for the transaction) the reference count associated
with the outbound TCP connection is incremented.
If the reference count now has a value of one (i.e., the
first reference) the AfterLife timer is stopped;
it will be restarted when all transactions referencing
the TCP connection are complete
(see section 6.4.7 Terminating TCP Connections for more information
on the AfterLife timer functions).

6.3.2 TransactionProgressing

In this state, the transaction is in progress and the device is
processing the transaction. For non-terminating devices this means
that the message has been sent to the successor device and
the device is waiting on the successor TCP connection
for a transaction response, a NOTIFY message
or for KEEPALIVE messages (see section 6.4.6
entitled Maintaining Peer Connectivity).

If the device is a terminating device for the
transaction, the device beings
processing the transaction. For example, if the
transaction is a trail create and the device
is the ONC, it will initiate the allocation of the optical trail.
Upon completion of the of the event, the device progresses to the state
TransactionComplete.

Note that in this state, if there is no response within
the transaction time-out time, transaction
state is cleared. No messages are sent as a result of transaction
time-out by non Requester devices. The Requester device
sends out a QUERY message if its transaction time-out
timer expires.

6.3.3 TransactionComplete

The device enters the TransactionComplete state if it has
completed its processing and, in the case of non-terminating
devices, received a response from the successor device; or,
if an error condition per section 6.5 has occurred. A response
or NOTIFY message is now sent to the predecessor device.

When a transaction message is received which completes
a transaction (i.e., a NOTIFY, Acknowledge or Response
message), the TCP connections' reference count is decremented
and the TCB is deallocated.
If reference count is now 0 for the TCP connection that
was initiated to the successor device, the AfterLife timer is started.

Note that if a transaction (NOTIFY or Acknowledgement) message is
received for a transaction for which a device no longer
has state, the message is silently discarded.

6.4 ODSI Inter-Device Transport

ODSI transactions require reliable and in-order delivery of messages. To satisfy these requirements, ODSI signaling control uses TCP as the inter-device transport mechanism for all control and notification messages. Multiple transactions may be performed across a single TCP connection.

6.4.1 Summary Of Operation

Upon initialization, all ODSI devices begin waiting on TCP port 1308 for incoming TCP connections from ODSI peers. After a TCP connection is established, the ODSI process waits on the TCP connection to receive incoming messages. Transactions are identified by a combination of the Trail ID and transaction type in the received message. An ODSI message is processed only after it is entirely received. The maximum message size is 4096 octets. All implementations are required to support this maximum message size.

Prior to relaying a transaction message to a peer device, a device checks for is a currently active TCP connection. If there is no TCP connections currently active, the device initiates a new TCP connection to the peer device (i.e., the successor device in the transaction's Control Points field). For the most part, a device will only have to initiate a new TCP connection when the device has received an event or a transaction message denoting that a new transaction has been initiated.

A TCP connection will remain active while all transactions that are using it are in progress. The lifetime of TCP connection depends a configuration parameter (AfterLife time) which defines the amount of time the connection will remain active after the last transaction using the connection has been completed. The default value for AfterLife timer is 60 seconds. For example, a device acting as a Trail Control Head may use the default value for TCP connections to devices acting as the Trail Control Tail but may want to keep a permanent TCP connection open with the device acting as the ONC (which may be it's local optical switch).

After the TCP connection is established, the initiator and peer exchange KEEPALIVE messages. A KEEPALIVE message includes a PeerDead time, which is the time period that the peer device shall wait to receive a message before transitioning the device to the down state. Devices should send at least one KEEPALIVE or transaction messages per PeerDead time interval.

Upon expiration of the AfterLife timer the initiator of the TCP connection sets a bit in its KEEPALIVE message informing the peer of its intent to shut down the TCP connection. The initiator closes the TCP connection after receiving an acknowledgment of its intent to shut down the TCP connection (see section 6.4.7). This mechanism is in place so that the peer device can choose its course of action (i.e, initiate a new TCP connection), in the event that it has just sent or is planning on sending the first message of a new transaction on the to-be-terminated TCP connection.

KEEPALIVEs are mandatory to send at the beginning and the end of a TCP session only. All other KEEPALIVEs are optional.

KEEPALIVE messages can be used to insure that TCP connections remains active. Since there are no explicit acknowledgments of messages and TCP is a reliable transport, by receiving messages, a peer can be assured that the messages that it has sent to its peer have been received.

Note that it is valid for a device to have two simultaneous TCP connections with a peer device at one time; one initiated by the device to its successor device and one initiated by the device's peer to the device itself. Each TCP connection has its own KEEPALIVE messages and timers. A device should use the same TCP connection for a single transaction.

6.4.2 ODSI Peer Table

ODSI represents it's inter-device TCP connectivity state with a peer table. An entry in the table is indexed using the peer device's IP address. The peer table contains the TCP connection identifiers, the state of the TCP connections and the timers associated with the TCP connections.

Note that multiple transactions may reference a single TCP connection. To facilitate the interactions between the TCP connection and the transactions referencing the connection, the transactions which reference the TCP connection must be accessible.

The key elements of an entry in this table are as follows.

- The IP address of the peer device. This is used as the index for this table entry and as the destination address for initiating connections to and for accepting connections from a peer device.

- TCP connection identifiers. There can be at most two TCP connections with the peer device; one initiated by the device itself (the outbound connection) and one initiated by the peer device (the inbound connection). The originator of the connection is responsible for terminating the connection. Each connection has an associated reference count which indicates when the connection no longer has transactions referencing it.

- The state of each a TCP connection. A connection may be in one of the following states.

  Down
  : The connection is down, which is each ODSI device's initial state.

  Connect
  : The device is in the process of establishing a TCP connection with the peer device.

  Established
  : The TCP connection has been established with the peer device. In this state the peering devices are exchanging KEEPALIVE and transaction messages.

Shutdown
The device is attempting to shutdown the connection with the peer device or (if the connection has been initiated by the peer device) the initiating device has notified this device of its intention to shut down the connection.

o ConnectFail timer
The timer used to determine that a TCP connection cannot be opened to a peer device. When a device initiates a TCP connection to its successor device (the successor device is found in the message's Trail Control Route TLV), it starts the ConnectFail timer. The exact value of the ConnectFail timer is a local matter, but should be sufficiently large to allow TCP initialization.

o KeepAlive timer
The timer used to send periodic KEEPALIVE messages to the peer device. There is one timer per TCP connection.

The default value for this timer is 30 seconds. The device sets the PeerDead timer value in its KEEPALIVE message to two times the KeepAlive value. When a KEEPALIVE or transaction message is sent to a peer, the timer is reset to the KeepAlive timer value. See section 6.4.1 for more details.

When the KeepAlive timer value is set to 0, the device should not send out the KeepAlive message to its peer.

o PeerDead timer
The timer used to determine that a TCP connection is no longer active. There is one dead timer per TCP connection. Upon expiration of a PeerDead timer, the connection is transitioned to the Down state. A PeerDead timer is reset to its time out value when a transaction or KEEPALIVE message is received.

If the PeerDead timer value is 0, it should never time-out.

The default value for this timer is 60 seconds.

Note that a TCP connection is timed out using the default PeerDead timer value when a KEEPALIVE is not received. This covers the following cases:

- A KEEPALIVE is not received by the peer device after accepting the TCP connection

- A KEEPALIVE or NOTIFY is not received by the initiator after sending the initial KEEPALIVE

- A KEEPALIVE is not received by the initiator reflecting the GoodBye flag

- The TCP connection has not been closed by the initiator after the peer device has responded to the GoodBye flag

- AfterLife timer
  This timer is started when the outbound connection's reference Count reaches zero. When the timer expires, the TCP connection will be shut down. Note that the timer may be set to infinity (0x0) to denote that the connection should remain permanent.

6.4.3 Using existing TCP connections

After a device has completed processing a transaction message it is ready to relay the message to its successor device.

- If there is a TCP connection which is in the Connect state the transaction waits for the connection to be established before it can relay the message. The reference count associated with the TCP connection is incremented.

- If there is a TCP connection which is in the Established state, the device uses it to relay the message to its successor device. If the transaction message requires a response (i.e., it is a request or a query message), the reference count associated with the TCP connection is incremented. If the reference count now has a value of one, the AfterLife timer it is stopped.

- If there is a TCP connection which is in the Shutdown state, the device schedules the connection to be re-established after the shutdown procedure is complete and follows the procedures for Initiating TCP Connections below. The transaction remains in the state TransactionConnect.

- If there is no TCP connection available, the device follows the procedures for Initiating TCP Connections below. The transaction remains in the state TransactionConnect.

6.4.4 Initiating TCP Connections

If there is no currently active TCP connection when the device is ready to relay the message to the successor device, a TCP connection is initiated. ODSI devices use TCP port 1308 for establishing its connections. Note that it is common practice for devices (see [Ref7]) which use TCP as a transport to use a loop-back IP interface to receive TCP connection request. Use of the loop-back IP interface makes the device resilient to the failure of any single interface.

When the connection initiated, the TCP connection is placed in the state Connect and the ConnectFail timer is started. The exact value of the ConnectFail timer is a local matter, but should be sufficiently large to allow TCP initialization. The transaction remains in the state TransactionConnect.

Upon notification of a successful TCP connection setup:

- The device initializes the PeerDead timer to the default value (60 seconds). The value will be set to the PeerDead timer value found in the KEEPALIVE message upon receiving the peer's first KEEPALIVE message.

- A KEEPALIVE message is immediately sent to the peer device to inform the peer of its PeerDead timer value.

- The device begins waiting to receive messages on the newly opened TCP connection.

- The processing of all TCBs that are dependent on the TCP connection is continued upon receiving the KEEPALIVE message from the peer device. At that point the pending messages are relayed to the successor device and the transaction state is progressed to TransactionProgressing.

- If the transaction (in the TransactionProgressing state) message requires a response, the reference count associated with the TCP connection is incremented. If the reference count now has a value of one, the AfterLife timer is stopped.

If the TCP connection cannot be opened to the successor device, the device follows the notification procedures (specified in section 6.5) including the error code, "TCP Open Failure" in the NOTIFY message.

6.4.5 Acquiring ODSI Peers

The ODSI process acquires ODSI peers by listening for (on port 1308) and accepting TCP connections. When the connection succeeds, the device initializes the peer table entry for the inbound TCP connection, sends a KEEPALIVE message to its peer, sets its PeerDead timer to the default PeerDead timer value, and sets the TCP connection state to Established.

6.4.6 Maintaining Peer Connectivity

While the TCP connection is active, the initiator and peer exchange KEEPALIVE messages; KEEPALIVE messages are sent by both sides of the TCP connection. KEEPALIVE messages insure that the TCP connection is currently active. Since there is no explicit acknowledgment of messages, and TCP is a reliable transport, by receiving KEEPALIVE messages, a peer can be assured that the messages that it has sent to its peer have been received. ODSI devices should use the same TCP connections for all exchanges associated with a single transaction.

KEEPALIVE messages include the PeerDead time period that the remote device will use to declare the sender of the KEEPALIVE to be unreachable. The device must send a KEEPALIVE or transaction message within the PeerDead time interval; when a KEEPALIVE message or a transaction is received the PeerDead timer is reset to the PeerDead time value.

The default value for the KeepAlive timer is 30 seconds. The device sets the PeerDead timer value in its KEEPALIVE message to

two times the KeepAlive value. When a KEEPALIVE or transaction message is sent to a peer, the timer is reset to the KeepAlive timer value.

The default value for the PeerDead time is 60 seconds. When a device receives an KEEPALIVE message or any other transaction message, it restarts the PeerDead timer using the PeerDead time found in the last KEEPALIVE message. A device assumes the PeerDead timer is the default value upon initialization. The PeerDead time can be configured on a per peer device bases.

If the PeerDead timer expires or a TCP connection is prematurely terminated the initiator device follows the notification procedures specified in section 6.5. A bulk notification messages is originated which include the Message IDs of the transactions using the TCP connection and indicating the error code TCP Failure.

The KEEPALIVE message contains an authentication field which is used to authenticate the peer relationship. If the peer receives a KEEPALIVE message with valid authentication, it responds with one KEEPALIVE message. If the received KEEPALIVE does not include a valid authentication key, the device follows the notification procedures (specified in section 6.5.3).

When a transaction message is received which completes a transaction (i.e., a notification or acknowledgment message) the TCP connection's reference count is decremented. If reference count is now 0, the AfterLife timer is started.

6.4.7 Terminating TCP Connections

To maximize the utilization of TCP connections, connections are kept active for a period of time after all transactions that are using them are completed. The period following the completion of the last transaction using a TCP connection is known as the AfterLife time. The AfterLife time is a per-peer configured value that defaults to the PeerDead time value. Note that a device may configure the AfterLife time to be infinite (e.g., in the case that a device wants to have its connection to the ONC be permanent).

Under normal conditions, the initiator of the TCP connection terminates the TCP connection with its peer TCPAfterLifeTime seconds after the last transaction using the TCP connection has completed.

The shutdown process for the connection initiator is as follows.

- Upon expiration of the AfterLife timer the initiating device sets the GoodBye flag the next outbound KEEPALIVE message.
- The TCP connection is transitioned to the state Shutdown.
- The device continues to receive messages on the TCP connection and begins waiting for acknowledgment of its intent to shut down the TCP connection (i.e., sees the GoodBye flag set in the peer's KEEPALIVE message).
- Upon receiving the acknowledgment of its

intent to shut down the TCP connection
the device closes the TCP connection and sets
the connection state to Down.

The shutdown process for the remote peer (the connection receiving side) is as follows.

- o The device checks for the GoodBye flag in all received KEEPALIVE message. Until being notified of intent to shutdown the connection by the connection's initiator, the device continues to send transaction and KEEPALIVE messages as needed.

- o When the device is notified of the initiator's intent to shutdown the connection, the device transitions the TCP connection to the Shutdown state and sends an KEEPALIVE message to the peer with the GoodBye flag set in the KEEPALIVE. At this point, no messages will be sent on the TCP connection.

- o Next, the device begins waiting for the connection initiator to terminate its TCP connection, at which point the connection is transitioned to the Down state.

6.5 Notification Procedures

Notification procedures describe the origination of NOTIFY messages for the following functions.

1) An ODSI transaction request has failed due to the mal-formed or illegal contents of a transaction message.

2) There has been a TCP establishment failure.

3) During the operation stage, a trail has failed. and restoration could not be performed (e.g., trail is unprotected or restoration failure). Reasons for such a failure include a cable cut, fiber cut etc.

4) The device's peer has initiated notification due to an error in a peer-to-peer message exchange (as opposed to a transaction message exchange).

5) The Trail ID is being sent to the Trail Control Requester.

In the cases where a transaction is in progress and the NOTIFY is indicating the failed transactions (cases 1 and 2), the NOTIFY messages include a list of the associated Message IDs (case 1 will include at most one Message ID). In the cases where there is an operational failure (case 3) the NOTIFY includes the list of failed trails by including a list of associated Trail ID. In case 5, the NOTIFY includes the newly allocated Trail ID as well as the Message ID. See Appendix A.7 for the NOTIFY message formats.

When the NOTIFY message is received by the Trail Control Requester (or the TCP connection has unexpectedly been terminated between the Trail Control Requester and its peer device) and if the result of the transaction is uncertain, the Trail Requester initiates a trail query to the ONC (and perhaps head and tail) to see what state the transaction appears to be in; the device should log the event with a management notification. The device may originate a DELETE message to clean up the transaction state.

Obviously, a new NOTIFY message is never originated as the result of receiving an incorrect or malformed NOTIFY message. If the NOTIFY message specifies a trail failure, the Trail Control Requester should log a network management notification event.

The notification types, causes for these notifications and contents of the notification are listed below. The detailed message formats can be found in Appendix A.7.

6.5.1 Transaction Related Notification

The method for dealing with message content related or TCP failure related incomplete transactions (i.e., a transaction related notification) is as follows. A NOTIFY message is sent along the Control Points path to the Trail Requester.

When a transaction related NOTIFY message is received by a device from its peer, it relays the NOTIFY message to its predecessor, enters state TransactionCompleting clears the transaction state and deletes its TCB.

Since there may be more than one transaction attempted which references the same trail (using the same Trail ID) the Trail ID cannot be used to uniquely identify an error condition to a peer. Instead, a transaction related NOTIFY messages uses a list of message IDs to identify the message (and the transaction) that has resulted in an unsuccessful transaction or error condition.

A Message ID is a 32-bit valued used to identify a message. A Message ID has peer-to-peer significance; the device's predecessor (the sender of the message) allocates a locally unique Message ID for each transaction message that it relays to its peer device. As a NOTIFY message is relayed back along the Control Points path to the Trail Control Requester, the Message ID is swapped out for the one that is locally significant (i.e., has been allocated by) the predecessor device.

If the TCP connection to the device's predecessor has been closed, the device's transaction state is cleared and the TCB is deleted. A NOTIFY will still be received by the Trail Control Requester because the predecessor device will originate a NOTIFIY message which will be relayed to Trail Control Requester.

6.5.2 Network Related Notification

The method for dealing with a network related failure such as network restoration failure or unprotected trail failed due to cable cut or fiber cut etc. is as follows. The device that recognizes the failure condition sends a NOTIFY message to the predecessor. If there are multiple affected transactions or failed trails, the NOTIFY message will include multiple Trail ID TLVs.

A device should include the configuration parameters to disable this function and to limit the number of notification messages sent to the predecessor.

6.5.3 Peer Initiated Notification

Peer-initiated notification refers to notification due to an error in a peer-to-peer message exchange (e.g., PeerDead time, bad authentication field, etc. in a KEEPALIVE message), as opposed to a transaction message exchange. The procedures for peer-initiated notification are as follows. If an error is detected by the peer that initiated the peer-to-peer TCP connection, this peer terminates the TCP connection. If an error is detected by the peer that didn't initiate the peer-to-peer TCP connection, this peer sends a NOTIFY message to the offending peer. The offending peer then terminates the TCP connection (noting the error as appropriate, e.g., KEEPALIVE Parse Error or Illegal Value). Note that the error may also trigger transaction-related notification procedures. NOTIFY messages are not sent in response to KEEPALIVEs not being received after a connection has been established.

6.5.4 Trail ID Notification

When a CREATE message received and the device is the Trail Control Head or there is no Trail ID in the message and the device is the ONC, the device allocates a Trail ID and sends a NOTIFY message back to the Trail Control Requester indicating a Trail ID. The Trail Control Head inserts the Trail ID in the CREATE message before the message is relayed to the next device in the Control Points field. As with section 6.5.1 above (Transaction Related Notification) the Message ID is included in the NOTIFY to uniquely identify the transaction.

6.5.5 NOTIFY Message Contents

The NOTIFY message includes the following information needed by the device receiving the NOTIFY to identify the cause of the NOTIFY and process the trail state for the associated trails.

- o Message Type
  Identifies the message as being a NOTIFY message.

- o Message Length
  Specifies the cumulative length in octets of the message including the Message ID, Mandatory Parameters, and Optional Parameters.

- o Message ID
  32-bit value used to identify this message.

- o Integrity TLV (Optional)
  This field contains authentication information.

- o Reporting Device
  The IP address of the device reporting the failure condition. If the error is due to a TCP connection failure, the devices with the failed connection will be the Reporting Device (predecessor) and its successor (as determined by the Trail Control Points field). This field is implemented using the

ODSI Address TLV.

- o Trail Control Points TLV
  This entry consists of 1) the number of control devices in the transaction chain, 2) an ordered list of devices constituting the transaction chain for this transaction and 3) the pointer to the next entry to process the message.

- o Notification Reason TLV
  TLV The reason for sending this NOTIFY message. Notification codes and Notification Sub-codes used in the Notification Reason TLV are listed below.

- o Notification Associations TLV
  This optional TLV consists of a list of Trail IDs or Message IDs used to identifying a trail or set of trails associated with the NOTIFY message.

- o Vendor-Specific TLVs
  When present, this field contains vendor specific information.

6.5.6 Notification Codes

Identifies the reason for the notification. Notification Codes are as follows.

1: TCP Open Failure
A TCP connection with the device's peer cannot be opened. The requesters of the in-process transaction (as identified by their Message IDs) must be informed of the incomplete transactions. A NOTIFY message is sent to each requester (along the Control Points path) including a list of all outstanding transactions (each identified by its Message ID). The NOTIFY also includes the address of the failed device.

Notification Associations:
List of Message IDs

2: KEEPALIVE Parse Error
The device has detected an error while attempting to parse a message. The Notification Information field will be set to "Parse Error" and the first 16 octets of the unparsable string will be included. The Sub-code field contains the octet offset of the error from the beginning of the failed message. The message is sent to the peer device only. The result of the error is that the TCP connection will be terminated by the connection's initiator. If termination has resulted one or more incomplete transactions the procedures for a TCP Failure above are followed.

Notification Associations:
Not present

3: KEEPALIVE Illegal Value
The device has determined that the KEEPALIVE message it has received contains an illegal value. The Sub-code field further qualifies the failure code. Depending

on the Sub-code type, the Notification Information field will provide the illegal value.
The message is sent to the peer device only.
The result of the error is that the TCP connection will be terminated by the connection's initiator.
If termination has resulted one or more incomplete transactions the procedures for a TCP Failure above are followed.

Notification Associations:
Not present

4: Transaction Message Parse Error
The device has detected an error while attempting to parse a message. The Notification Information field will be set to "Parse Error" and the first 16 octets of the unparsable string will be included. The Sub-code field contains the octet offset of the error from the beginning of the failed message.

Notification Associations:
Message ID

5: Modify Detected
The device has determined that a modify-class transaction is currently in progress which conflicts with the modify message that has just been received.

Notification Associations:
Message ID

6: Transaction Illegal Value
The device has determined that the message it has received contains an illegal value. The Sub-code field further qualifies the failure code. Depending on the Sub-code type, the Notification Information field will provide the illegal value.

Notification Associations:
Message ID

7: Insufficient Resources
There are insufficient resources to honor the request. The Notification Information field includes a list of resource TLVs indicating the available resources. The list of available resources will be based on the resources that have been requested by the Trail Create or Trail Modify that cannot be fulfilled. For example, if the amount of bandwidth that has been requested from the network is not available, the ONC will indicate the maximum amount of bandwidth that can be obtained to the specified endpoint.

Notification Associations:
Message ID

8: Unknown Trail ID
An unknown Trail ID was specified in a request. The Sub-code field identifies which (1=first, 2=second, etc.) of the trail IDs is unknown, if the request includes multiple Trail IDs.

Notification Associations:

Message ID

9: Optical network internal failure.
The device has determined that the trail has failed due to an internal optical network failure. This case doesn't cover the case where the edge optical network device failed.

Notification Associations:
List of Trail IDs

10: Optical Network Hand-Off Link Failure
The device has decided that the trail has failed due to a failure of the physical link between user and network device. This case also covers the scenario where an edge optical network device failed. The exact failure case is specified by the failure Sub-code field.

Notification Associations:
List of Trail IDs

11: Trail ID Identification
A Trail Control Head or ONC is responding to a CREATE message and informing the requester of the Trail ID that will be used for the newly created trail.

Notification Associations:
Trail ID
Message ID

6.5.7 Notification Sub-Codes

The sub-code is used to further qualify a NOTIFY that was originated as the result of an illegal value or on optical network failure. Section 6.5.6 specifies the use of the sub-code and information field for each of the failure cases.

In the case of an illegal value, the Notification Information field includes the illegal value which may be a direct value as a 4-octet integer or a further nested TLV (e.g., User Group Authentication). Notification Sub-code values are as follows.

1: Incorrect Message Type

2: Incorrect Message Length

3: Unknown TLV Class-Num

4: Unknown TLV C-Type

5: Mandatory TLV missing

6: Illegal Trail ID

7: Unacceptable Trail User's Group

8: Unacceptable Trail User Group Authentication

9: Illegal Value In Trail Control Route

10: Illegal Bandwidth Size

11: Illegal Propagation Delay Value

12: Illegal Trail Head Port

13: Illegal Trail Tail Port

14: Head End Hand-Off Link Failure

15: Tail End Hand-Off Link Failure

16: Head End Edge Optical Device Failure

17: Tail End Edge Optical Device Failure

7.0 Configuration Parameters

The following items are needed to configure an ODSI device.

- Device's IP Address
  The IP address of the device. This is used for initiating for receiving TCP connection. It is common practice for devices (see [Ref7]) which use TCP as a transport to use a loop-back IP interface to receive TCP connection request as this makes the device more resilient to the failure of any single interface.

- Port and channel address information
  Ports are addressed using an IP address, a port index and if needed the channel number. The port index identifies a particular physical interface connecting into the Optical core. In IP MIBs, this is commonly an IfIndex.

- ConnectFail timer value
  There is a ConnectFail timer associated with a device's predecessor and successor device. The exact value of the ConnectFail timer is a local matter, but should be sufficiently large to allow TCP initialization. See section, 6.4 "Transaction Control Block" for more timer details.

- KeepAlive timer value
  The timer used to in consequence with the successor's ConnectFail timer to insure the viability of the TCP connection with the peer device. It is used to periodically send KEEPALIVE messages and to insure transaction messages are sent to the peer device while the peer is in the Established state.

  The default value for this timer is 30 seconds. The device sets the PeerDead timer value in its KEEPALIVE message to two times the KeepAlive value. When a KEEPALIVE or transaction message is sent to a peer, the timer is reset to the KeepAlive timer value. See section, 6.4 ODSI Inter-Device Transport for more details.

- TransactionDead timer value
  The TransactionDead timer is used to timeout a transaction when the network fails to complete the transaction. The exact value of the timer

is a local matter, but should be sufficiently large to allow the transaction to complete.

- o AfterLife timer value
  The timer used to tear down the TCP connection when the outbound connection's reference Count reaches zero. The exact value of the ConnectFail timer is a local matter, Note that the timer may be set to infinity (0x0) to denote that the connection should remain permanent. See section, 6.4.2 "ODSI Inter-Device Transport" for more timer details.

- o List Of User's Groups
  The per-port list of user's groups supported by the device. Each entry should include the user group ID, the port identifier and signature per user's group if required.

References

[Ref1] G. Bernstein, R. Coltun, J. Moy, A. Sodder and K. Arvind, "ODSI Functional Specification", ODSI Coalition, August 2000

[Ref2] B. Rajagopalan, et al. "IP over Optical Networks: A Framework", work in progress, IETF Internet Draft

[Ref3] Abould-Magd, O.S., et al. "Signaling Requirements at the Optical UNI", work in progress, IETF Internet Draft

[Ref4] Rosen, E., Viswanathan, A., Callon, R., "Multiprotocol Label Switching Architecture", work in progress, IETF Internet Draft

[Ref5] Bellcore Generic Requirements, GR-253-CORE, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, Issue 2, December 1995.

[Ref6] Boyle, J., R. Cohen, D. Durham, S. Herzog, R. Rajan and A. Sastry, "The COPS (Common Open Policy Service) Protocol", work in progress, IETF Internet Draft

[Ref7] Rekhter, Y., Li, T., "A Border Gateway Protocol 4 (BGP-4)", RFC1771, March 1995.

[Ref8] Ashwood-Smit, P., et al., "Generalized MPLS - Signaling Functional Description", http://search.ietf.org/internet-drafts/draft-ietf-mpls-generalized-signaling-00.txt

[Ref9] B. Fox and B. Gleeson, "VPN Identifiers," RFC 2685, Sepember 1999.

[Ref10] S. Chaudhuri, G. Hjalmtysson and J. Yates, "Control of Lightpaths in an Optical Network," OIF2000.004.0

[Ref11] F. Baker, B. Lindell and M. Talwar, "RSVP Cryptographic Authentication," RFC2747, January 2000

[Ref12] G. Bernstein, R. Coltun, J. Moy, A. Sodder and K. Arvind,

"ODSI Service Discovery and Address Registration",
ODSI Coalition, April 2000

[Ref13] ITU-T Recommendation G.707 Network Node Interface for the Synchronous Digital Hierarchy (SDH), March 1996

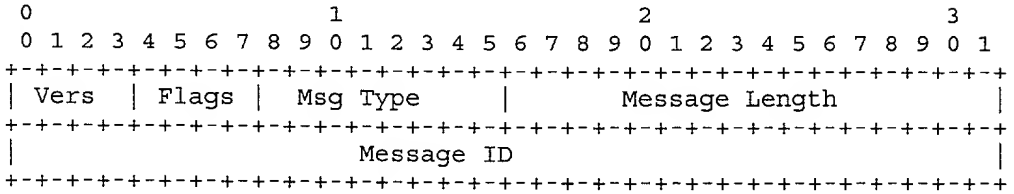
[Ref14] N. Ghani, Z. Zhang, L. Zhang, J. Fu and J. Moy, "COPS Usage for ODSI," ODSI Coalition, August 2000

[Ref15] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985

Appendix A: Message Formats

An ODSI signaling message consists of a common header, followed by a body consisting of a variable number of variable-length, typed "objects" (TLVs). The following subsections define the formats of the common header, the standard TLV header, and each of the ODSI signaling message types. Section 5.0 "Message Contents" describes the contents fields and usage.

Appendix A.1: Common Message Header



The fields in the common header are as follows:

Vers: 4 bits
Protocol version number. This is version 1.

Flags: 4 bits
0x02-0x08: Reserved

GoodBye flag: 0x01
The GoodBye flag is used by the originator of the TCP connection to inform the passive device (at the other side of the connection) that the connection is being terminated. The passive device acknowledges the reception of the KEEPALIVE flag by setting the GoodBye flag in its KEEPALIVE message. See Section 6.4.7 for more information.

Msg Type: 8 bits

1 = KEEPALIVE Message

2 = CREATE Message

3 = CREATE ACK Message

4 = MODIFY Message

5 = MODIFY ACK Message

6 = DELETE Message

7 = DELETE ACK Message

8 = QUERY Message

9 = QUERY RESPONSE Message

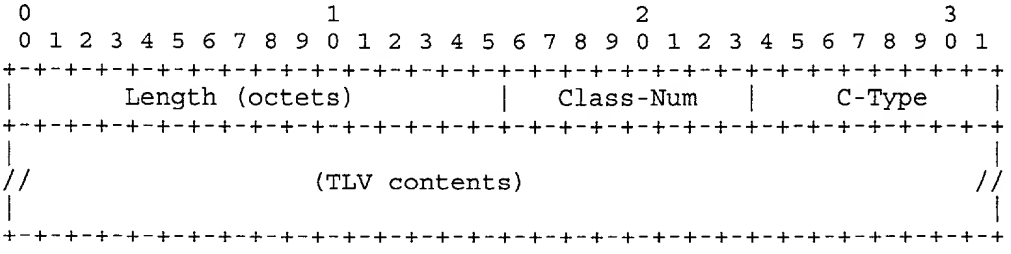
10 = NOTIFY Message

Message Length: 16 bits
The total length of this message in octets, including the common header and the variable-length TLVs that follow. The maximum length of an ODSI signaling message is 4096 octets.

Message ID: 32 bits
32-bit value used to identify this message.

Appendix A.2: TLV Formats

Every TLV consists of one or more 32-bit words with a one- word header, with the following format:



A TLV header has the following fields:

Length
A 16-bit field containing the total TLV length in octet. Must always be a multiple of 4, and at least 4.

Class-Num
Identifies the TLV class. Values are defined below.

C-Type
TLV type, unique within Class-Num. Values are defined below.

The Class-Num and C-Type fields may be used together as a 16-bit number to define a unique type for each TLV.

The high-order bit of the Class-Num is used to determine what action a node should take if it does not recognize the Class-Num of a TLV:

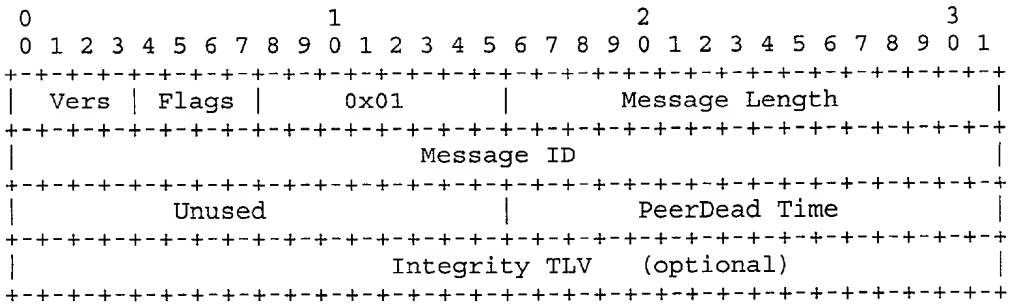
- o Class-Num = 0bbbbbbb
  The entire message should be rejected and an "Unknown TLV Class" error returned.

- o Class-Num = 1bbbbbbb
  The node should ignore the TLV, neither forwarding it nor sending an error message.

Appendix A.3: Message Definitions

Message are described in section 4.0. TLVs in messages are mandatory unless indicated otherwise. If more instances of a TLV appear in a message than the message format permits, the additional TLVs are ignored. The Integrity TLV, if present, is the first TLV to appear in any message.

A.3.1 KEEPALIVE Messages

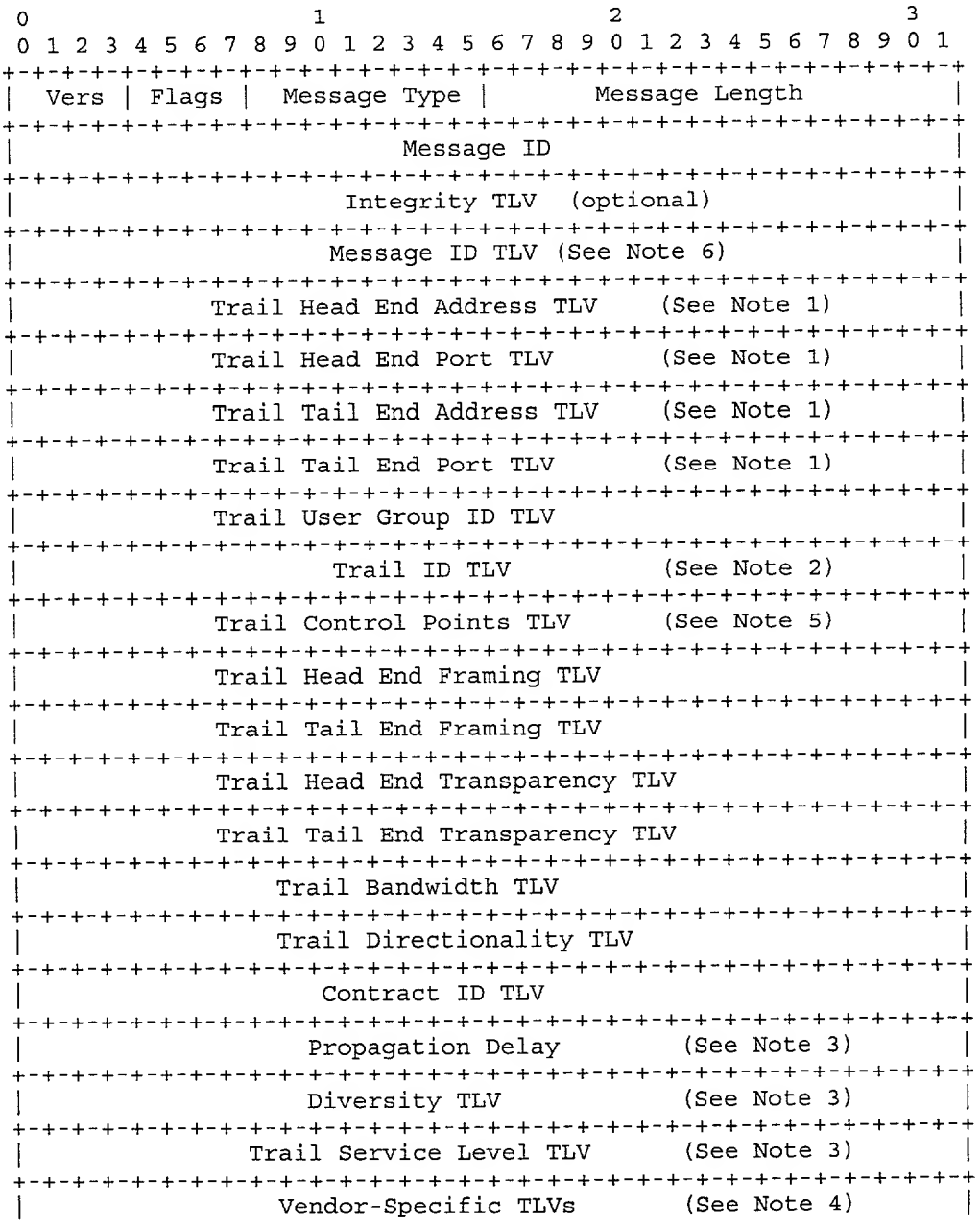


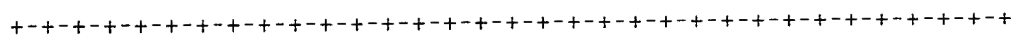
Flags
GoodBye flag: 0x01
The GoodBye flag is used by the originator of the TCP connection to inform the passive device (at the other side of the connection) that the connection is being terminated. The passive device acknowledges the reception of the KEEPALIVE flag by setting the GoodBye flag in its KEEPALIVE message. See Section 6.4.7 for more information.

PeerDead Time
The time in seconds that the device receiving the KEEPALIVE message will let elapse without receiving a transaction or KEEPALIVE message before it declares the peer as down.

A.3.2 CREATE (ACK), MODIFY (ACK) and QUERY RESPONSE Messages

The following defines the set of common TLVs used for CREATE, CREATE ACK, MODIFY MODIFY ACK and QUERY RESPONSE messages. The optional parameters and special case parameters are noted below. The Message Type field is assigned as noted in section A.1 above. The TLVs are defined in the following sections. Note that for a QUERY RESPONSE message, the Qualifier field of Trail ID will contain the state of the trail.





Option notes:

Note 1 -May be added to the CREATE by the Trail Control Requester, or the Trail Control Head or Tail. Not present in a MODIFY but is present in CREATE and MODIFIY ACKs and QUERY RESPONSE messages.

Note 2 -Not present in the exchange between Trail Requester and Trail Head in Trail Create Request. Present in all other cases.

Note 3 -Optional for a CREATE message. It is present for all other message only if was included in the original CREATE message.

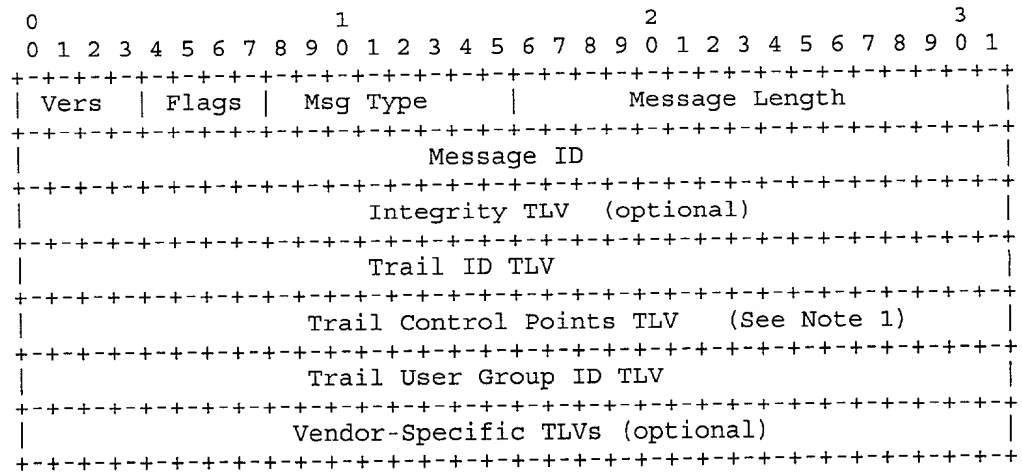
Note 4 -May be present in all message types.

Note 5 -Not needed in the ACK messages.

Note 6 -Only needed in the Query Response message, which contains the message ID generated by the upstream node for the corresponding QUERY

A.3.3 DELETE and DELETE ACK Messages

The following defines the set of TLVs used for the DELETE and DELETE ACK messages. Message Type fields are designated in section A.1 above.



Option notes:

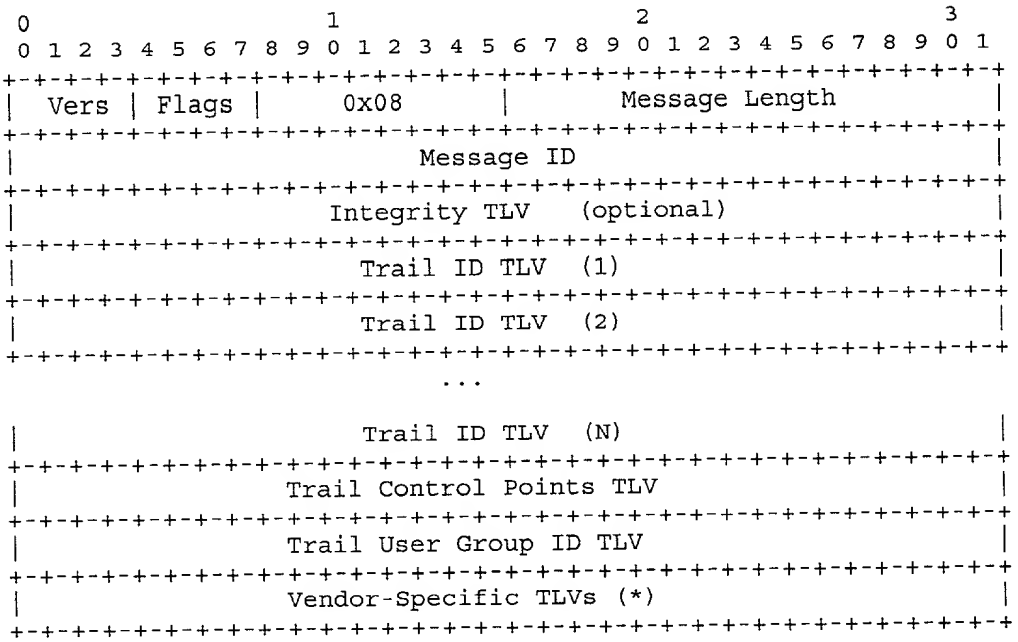
Note 1 -Not needed in the DELETE ACK message.

A.3.4 QUERY Message

The following defines the message format of a QUERY message. Note that the QUERY may request the status of several trails at once; several QUERY RESPONSE messages may be returned as the

result of a single QUERY.

The Trail Control Points TLV will typically contain two entries; one for the Trail Control Requester and one for either of 1) the ONC, 2) the Trail Control Head or 3) the Trail Control Tail.

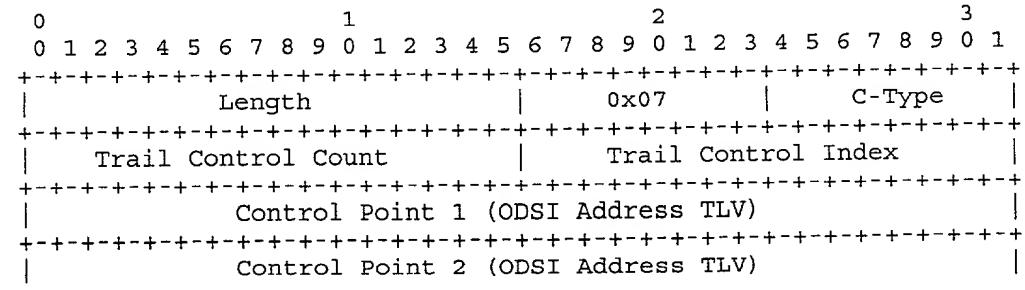


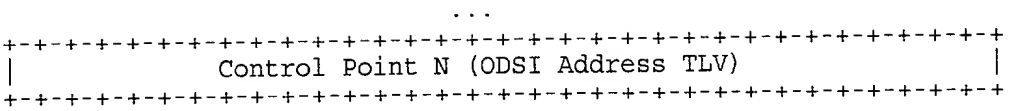
A.4 Message and Transaction Control TLVs

Descriptions of most of these fields are found in section 5.1. The five fields (Vers, Flags, Message Type, Message Length and Message ID) that are part of the Common Message Header are found in Appendix A.1.

A.4.1 Control Points TLV

The trail Control Points TLV consists of a count of entries, an index (which is used reference the current control points during the transaction process), and a list of control points which are encoded using ODSI Address TLVs. Trail Control Index should be 1-based. ODSI Address TLVs C-Type field denotes the control function of the entry. Note that the Trail Control Requester entry must always be present and be the first entry.





Trail Control Points TLV: Class-Num = 0x07, C-Type = 0x01

Trail Control Count: 2 octet
Number of hops in the transaction chain

Trail Control Index: 2 octet
Index used to identify the next hop to process the message

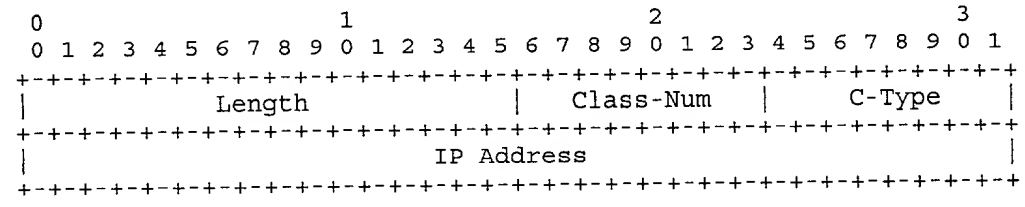
Control Point List
List of control points encoded using the ODSI Address TLV

A.5 Endpoint Identification TLVs

Descriptions of TLV definitions for endpoint identification are found in section 5.2.

A.5.1 ODSI Address TLV

This TLV is used to identify endpoint addresses, control point addresses and by a NOTIFY to indicate the source of a failure. The C-Type field is used as discriminator.



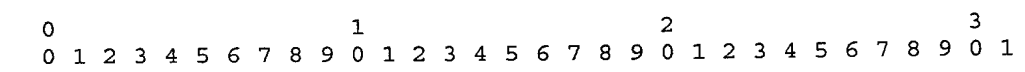
Length
This value is set to 4 for IPv4 addresses.

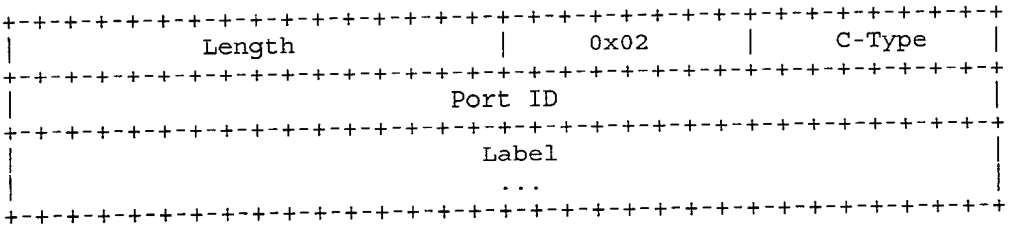
Class-Num and C-Type are assigned as follows:

IPv4 Trail Head End Address:	Class-Num = 0x01, C-Type = 0x01
IPv4 Trail Tail End Address:	Class-Num = 0x01, C-type = 0x02
IPv4 Reporting Device Address:	Class-Num = 0x01, C-Type = 0x03
IPv4 Trail Control Requester:	Class-Num = 0x01, C-Type = 0x04
IPv4 Trail Control Head:	Class-Num = 0x01, C-Type = 0x05
IPv4 Trail Control Tail:	Class-Num = 0x01, C-Type = 0x06
IPv4 ONC Address:	Class-Num = 0x01, C-Type = 0x07

IP Address: 32 bits
The IPv4 address of trail end point

A.5.2 Endpoint Port TLV





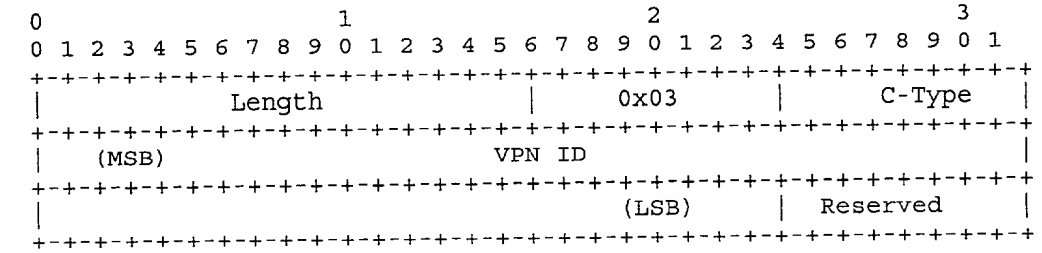
Class-Num and C-Type are assigned as follows:

Trail Head End Port:	Class-Num = 0x02, C-Type = 0x01
Trail Tail End Port:	Class-Num = 0x02, C-Type = 0x02

Port ID: 4 octets
Port ID (typically an RFC2233 ifIndex) associated with the port on which the endpoint resides.

Label:
This field uses the same format used for the label field in the Generalized Label object described in [Ref8] section 3.2.1. One or more labels, along with the bandwidth specified in the Bandwidth TLV identify the connection end point.

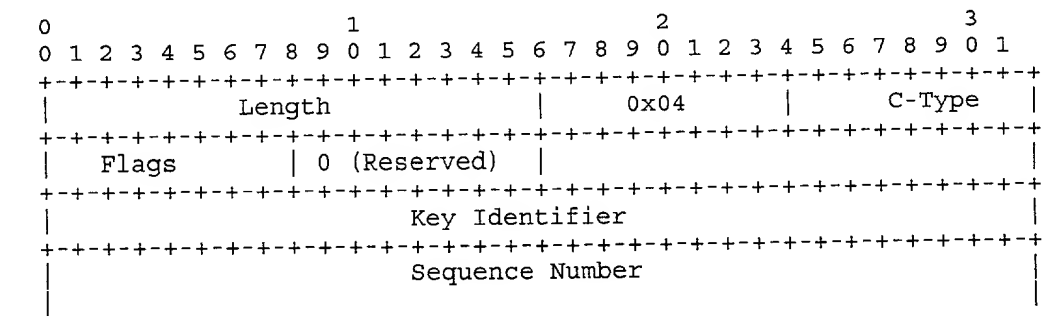
A.5.3 Trail User Group ID TLV

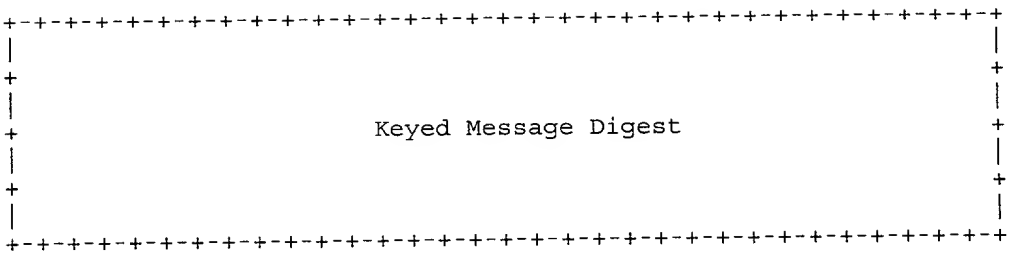


Trail User Group ID TLV: Class-Num = 0x03, C-Type = 0x01

VPN ID: 7 octets
User group ID encoded as a 7-octet VPN ID [Ref9].

A.5.4 Integrity TLV

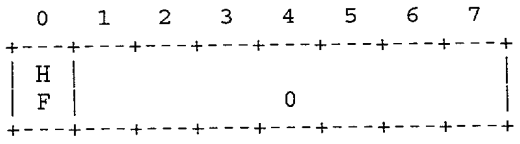




Integrity TLV: Class-Num = 0x04, C-Type = 0x01

Flags: An 8-bit field with the following format:

Flags



Currently only one flag (HF) is defined. The remaining flags are reserved for future use and MUST be set to 0.

Bit 0: Handshake Flag (HF) concerns the integrity handshake mechanism. Message senders willing to respond to integrity handshake messages SHOULD set this flag to 1 whereas those that will reject integrity handshake messages SHOULD set this to 0.

This ODSI version doesn't support integrity handshake mechanism, so Bit 0 should be set to 0.

Key Identifier: An unsigned 48-bit number that MUST be unique for a given sender. Locally unique Key Identifiers can be generated using some combination of the address (IP or MAC or LIH) of the sending interface and the key number. The combination of the Key Identifier and the sending system's IP address uniquely identifies the security association.

Sequence Number: An unsigned 64-bit monotonically increasing, unique sequence number.

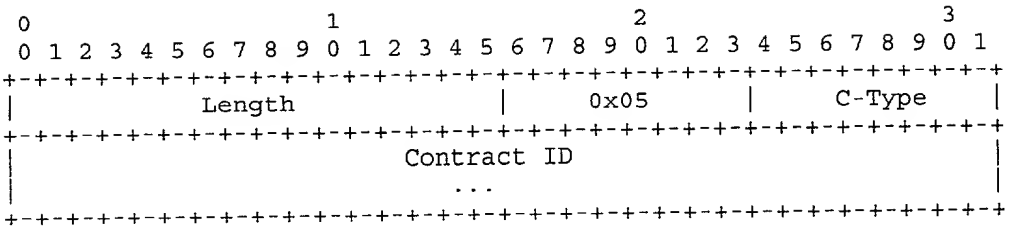
Sequence Number values may be any monotonically increasing sequence that provides the INTEGRITY object [of each ODSI message] with a tag that is unique for the associated key's lifetime.

Keyed Message Digest: The digest MUST be a multiple of 4 octets long.

See section 5.1 for a detailed description of the Integrity TLV.

A.6 Bandwidth Characteristics TLVs

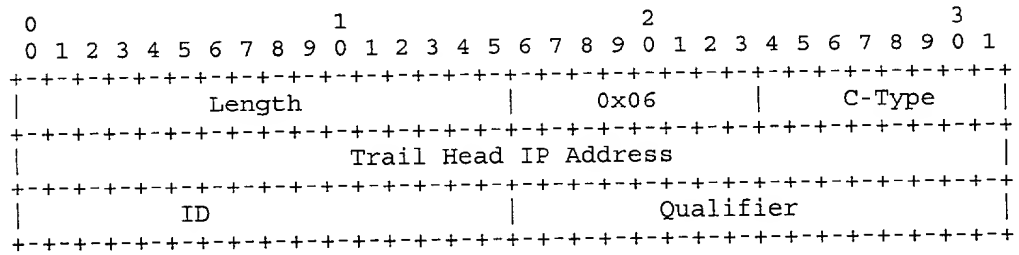
A.6.1 Contract ID TLV



Contract ID TLV: Class-Num = 0x05, C-Type = 0x01

Contract ID:
Contract ID encoded as a 4-octet aligned string of octets.

A.6.2 Trail ID TLV



Contract Trail ID TLV (Class-Num = 0x06, C-Type = 0x01)

Trail Head IP Address:
An IP address associated with the trail head.

ID:
Locally ID assigned 16-bit integer assigned to this trail
by the Trail Control Head or ONC.

Qualifier:
The qualifier field is not part of the Trail ID itself
but is a part of the TLV that qualifies a trail.
It is used when the Trail ID is nested in other TLVs or
is in specific message types.

When used with the Diversity TLV the Qualifier is
assigned one of the following values which represent
a diversity type.

1: linkDiverse

2: nodeDiverse

3: srlgDiverse

When used in a QUERY RESPONSE message the Qualifier is
assigned one the following values which represent
the state of the trail.

4: Trail Active

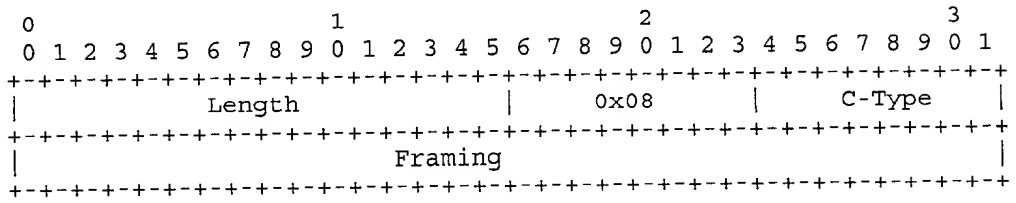
5: Trail Pending (waiting to be brought up)

6: Trail Inactive

7: Trail Unknown (there is no record of this trail)

At all other times the Qualifier value is 0.

A.6.3 Framing TLV



Class-Num and C-Type are assigned as follows:
Trail Head End Framing TLV: Class-Num = 0x08, C-Type = 0x01
Trail Tail End Framing TLV: Class-Num = 0x08, C-Type = 0x02

Framing: 4 octets
The format of the signal between user and network devices

0x01: pdh

0x02: sonet

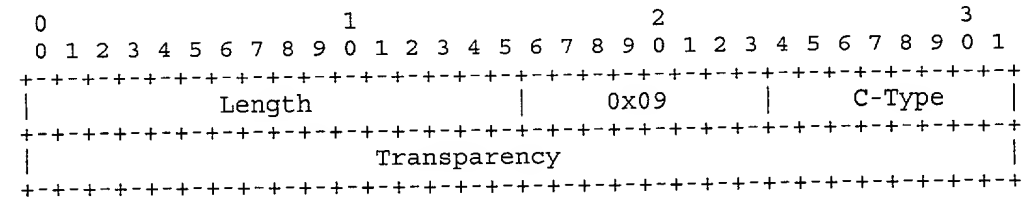
0x03: sdh

0x04: digitalWrapper

0x05: lanEthernet

0x06: wanEthernet

A.6.4 Transparency TLV



Class-Num and C-Type are assigned as follows:

Head End Transparency: Class-Num = 0x09, C-Type = 0x01
Tail End Transparency: Class-Num = 0x09, C-Type = 0x02

Transparency: 4 octets
The portion of the signal that is available for user data traffic.

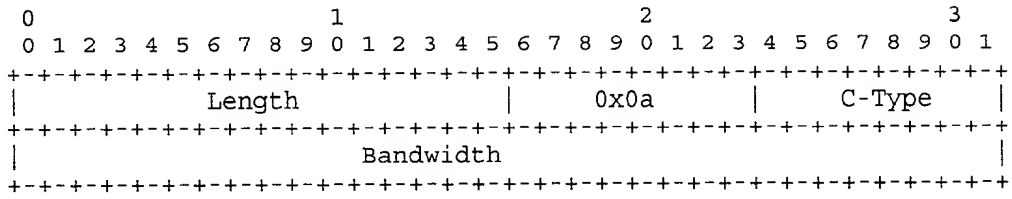
0x01: other

0x02: PLR-C

0x03: STE-C

0x04: LTE-C

A.6.5 Bandwidth TLV

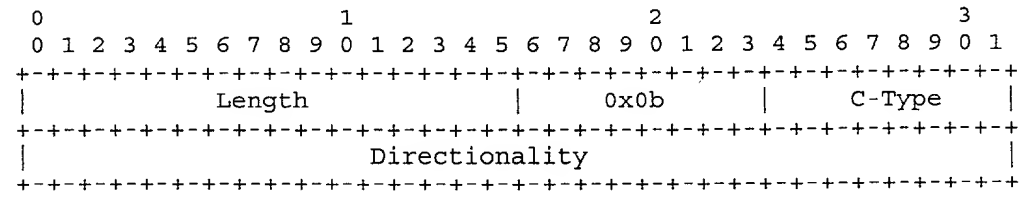


Bandwidth TLV: Class-Num = 0x0a, C-Type = 0x01

Bandwidth: 4 octets
The bandwidth requested for the trail in bytes per second. The bandwidth is encoded using the IEEE standard for normalized single-precision floating point numbers [Ref15]. The following list shows encodings for some representative bandwidth values.

Signal Type	(Bit-rate)	Encoding (bytes/sec)
DS0	(0.064 Mbps)	0x45fa0000
DS1	(1.544 Mbps)	0x483c7a00
E1	(2.048 Mbps)	0x487a0000
DS2	(6.312 Mbps)	0x4940a080
E2	(8.448 Mbps)	0x4980e800
Ethernet	(10.00 Mbps)	0x49989680
E3	(34.368 Mbps)	0x4a831a80
DS3	(44.736 Mbps)	0x4aaaa780
STS-1	(51.84 Mbps)	0x4ac5c100
Fast Ethernet	(100.00 Mbps)	0x4b3ebc20
E4	(139.264 Mbps)	0x4b84d000
OC-3/STM-1	(155.52 Mbps)	0x4b9450c0
OC-12/STM-4	(622.08 Mbps)	0x4c9450c0
GigE	(1000.00 Mbps)	0x4cee6b28
OC-48/STM-16	(2488.32 Mbps)	0x4d9450c0
OC-192/STM-64	(9953.28 Mbps)	0x4e9450c0
10GigE-LAN	(10000.00 Mbps)	0x4e9502f9

A.6.6 Trail Directionality TLV



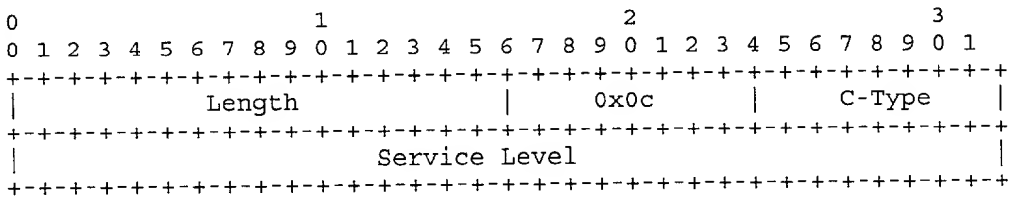
Directionality TLV: Class-Num = 0x0b, C-Type = 0x01

Directionality: 4 octets
Indicates if the trail is unidirectional or bidirectional.

0x01: unidirectional

0x02: bidirectional

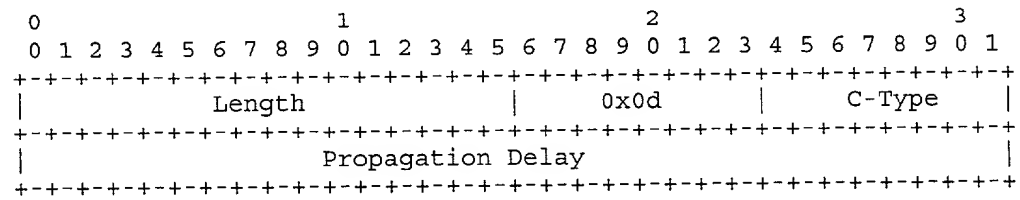
A.6.7 Trail Service Level TLV



Service Level TLV: Class-Num = 0x0c, C-Type = 0x01

Service Level: 4 octets
Service level requested for the trail. The values for service level are service provider-specific.

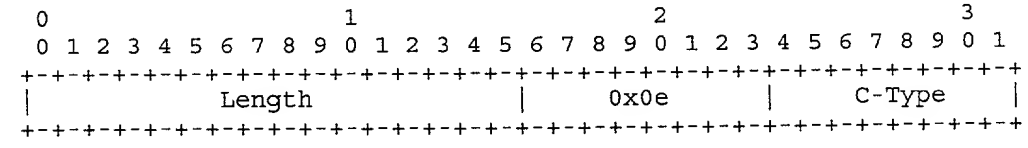
A.6.8 Trail Propagation Delay TLV

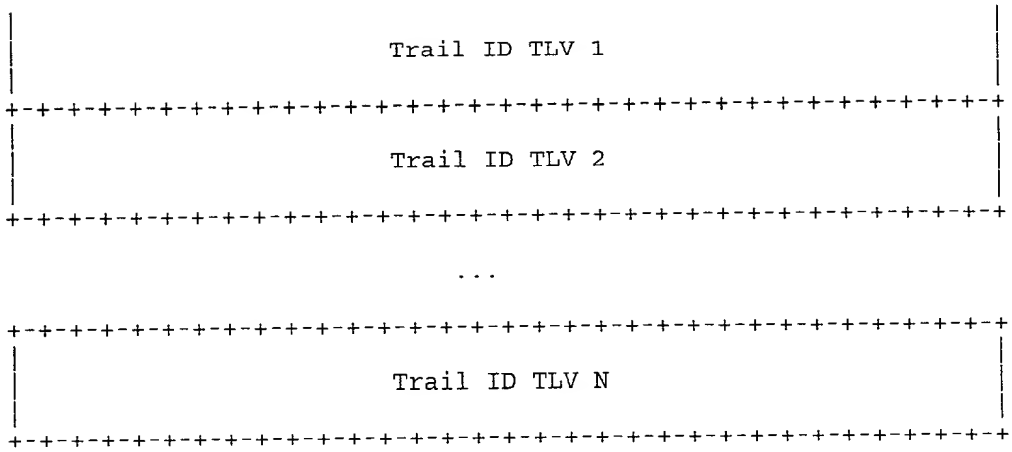


Propagation Delay TLV: Class-Num = 0x0d, C-Type = 0x01

Propagation Delay: 4 octets
The maximum amount of propagation delay in milliseconds to be tolerated on the trail. A value of 0 is interpreted to mean that the propagation delay is unspecified.

A.6.9 Diversity TLV





Diversity TLV: Class-Num = 0x0e, C-Type = 0x01

The Qualifier field in the Trail ID TLV defines the diversity type to be one of the following.

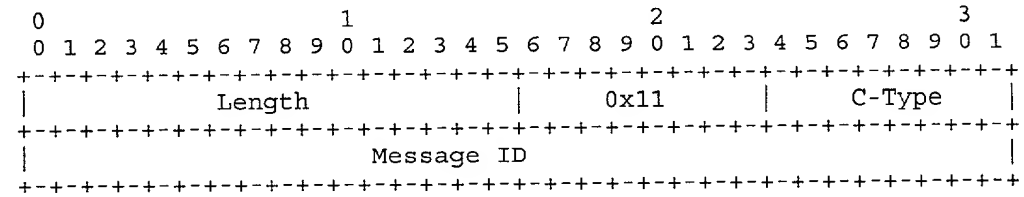
1: linkDiverse

2: nodeDiverse

3: srlgDiverse

A.6.10 Message ID TLV

Message ID TLV (Class-Num = 0x11, C-Type = 0x01)

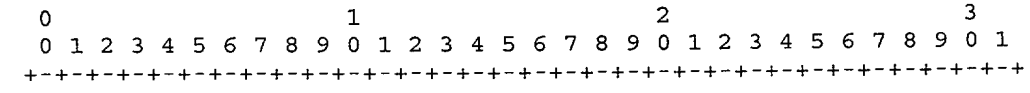


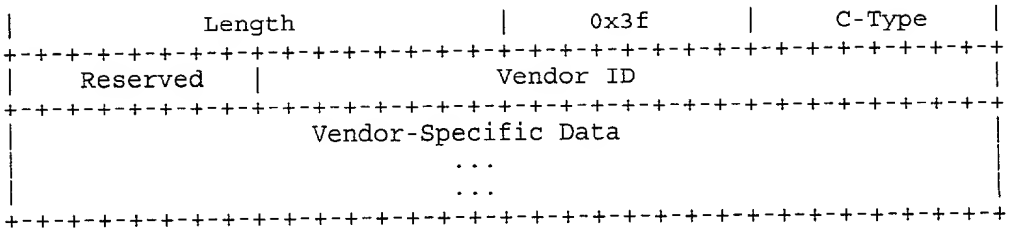
Message ID: 4 octets

A.6.11 Vendor-Specific TLV

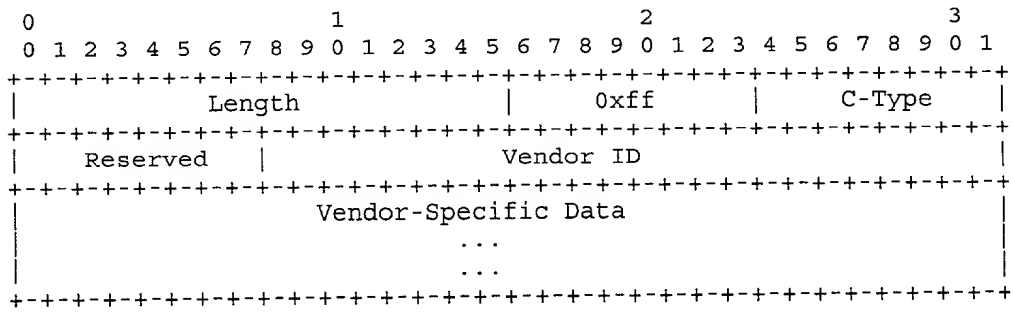
Two different Vendor-Specific TLVs are supported:

Vendor-Specific TLV 0: Class-Num = 0x3f, C-Type = 0x01





Vendor-Specific TLV 1: Class-Num = 0xff, C-Type = 0x01

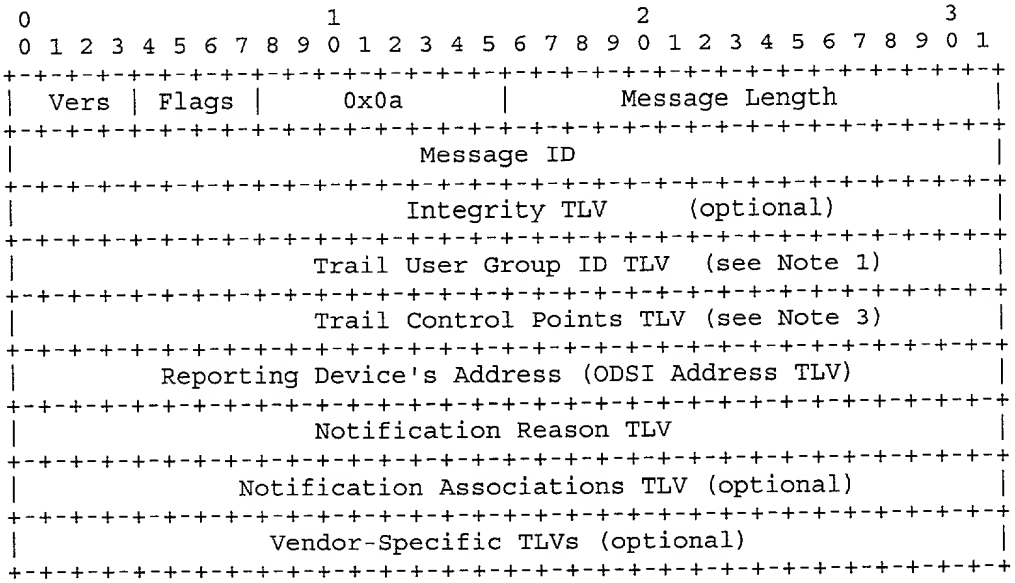


Vendor ID: 3 octets
  24-bit 802 vendor ID as assigned by the IEEE

Vendor-Specific Data:
  Defined by vendor. For example, a vendor could nest proprietary
  TLVs in this field.

A.7 NOTIFY Message Format

A.7.1 NOTIFY Message

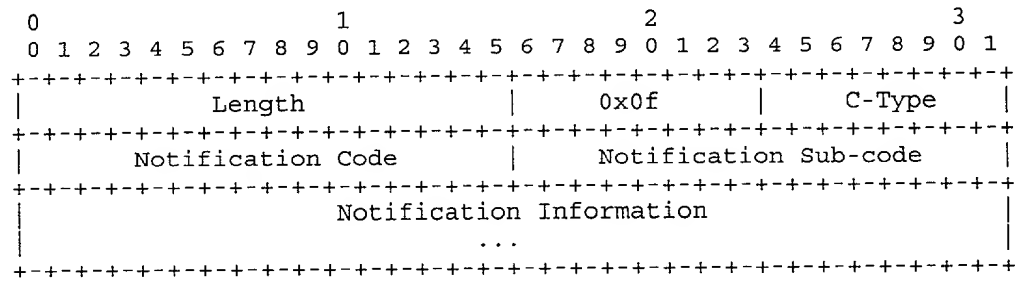


Note 1: The Trail User Group ID TLV is not present in peer-initiated (i.e., KEEPALIVE-related notifications). It is required in all other types of notifications.

Note 2: All the elements (Message ID TLV or Trail ID TLV) of the Notification Associations TLV shall pertain to the User Group specified by the Trail User Group ID TLV, if the Trail User Group ID TLV is present.

Note 3: Trail Control Points TLV is only needed in the Network Related Notification.

A.7.2 Notification Reason



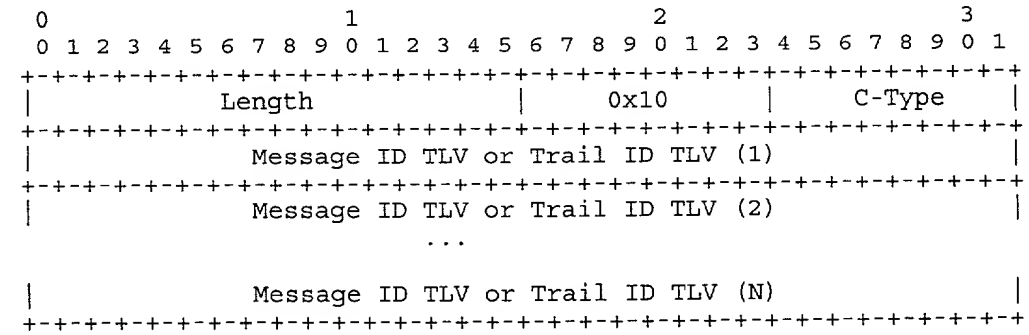
Notification Reason TLV: Class-Num = 0x0f, C-Type = 0x01

Notification Code: 16 bits
Denotes the reason for the notification.
See section 6.5.6 for specific values and usage.

Notification Sub-code: 16 bits
Qualifies the failure code further
See section 6.5.7 for specific values and usage.

Notification Information
Provides additional details helpful in interpreting the error condition. See sections 6.5.6 and 6.5.7 for more information.

A.7.3 Notification Associations

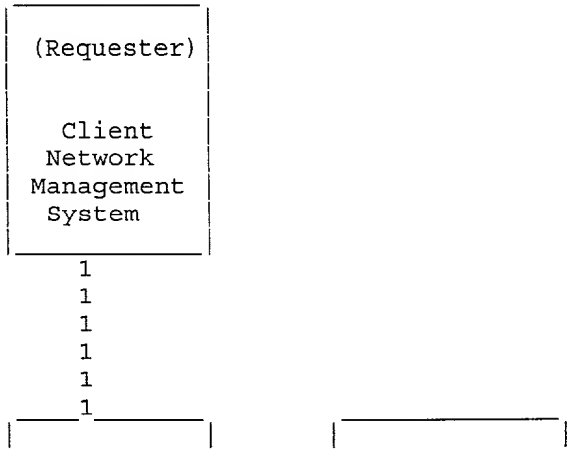


Notification Associations TLV: Class-Num = 0x10, C-Type = 0x01

Appendix B: Examples

Appendix B.1: Example 1

A management system for a client network requests additional bandwidth between edge nodes A and Z using the traditional control model.



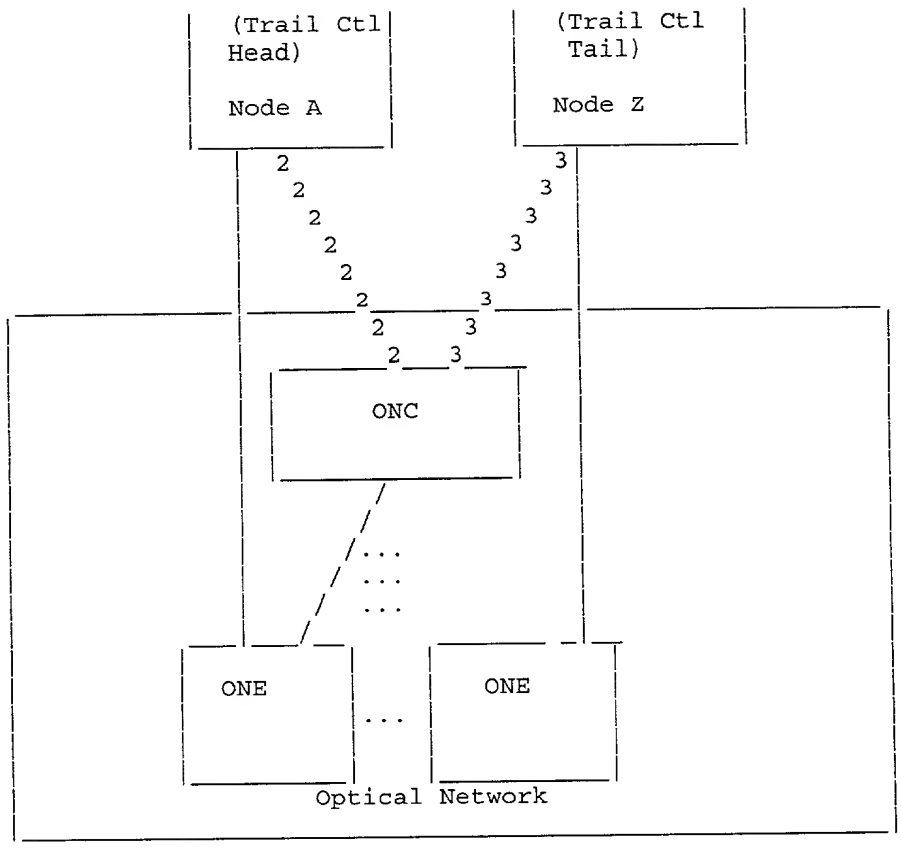


Figure 1

Description: The management system is the Trail Control Requester. It knows the ports on the edge nodes A (the Trail Control Head) and Z (the Trail Control Tail) that will be used for the connection. The Trail Control Requester will send a transaction message (1) to the Trail Control Head. The Trail Control Head assigns a unique Trail ID to the trail and sends the CREATE message (2) to the optical network via its interface to the Optical Network Element (ONE) connected to Edge Node A via the port specified in the CREATE message. This ONE forwards the CREATE message to the Optical Network Controller (ONC), which manages the connection across the optical network. Then the ONC sends a CREATE message (3) to edge node Z, the Trail Control Tail, via the ONE adjacent to it, and wait for the acknowledgement from node Z before setting up the connection across the optical network. After setting up the connection, the ONC will send a CREATE ACK back to the Trail Head, node A, and the Trail Head will send the create ack back to the Requestor with the Trail Id.

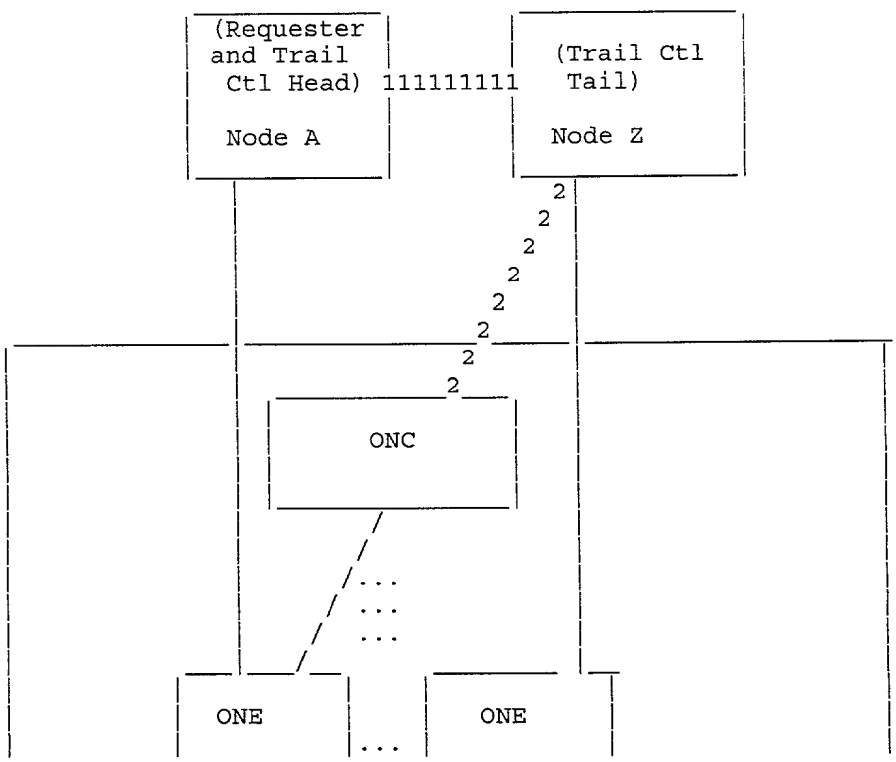
Comments: The edge devices in this scenario could be a variety of types of devices managed by the management system (who has taken on the role of the requester). They could be a pair of routers in an IP client network, a pair of ATM switches, or a pair of SONET/SDH ADMs in two separate metro networks. The management system could be a "real" management system, or a terminal interface to the Trail Control Head using the operator's intelligence control.

This scenario could be used before any direct communication is

possible between the Trail Control Head and Trail Control Tail, e.g. when first setting up the network. After there is a payload connection between A and Z, some of the bandwidth in that connection can be used for signaling to allow the other control model to be used.

Appendix B.2: Example 2

An edge node, A, in a client network requests additional bandwidth to another edge node,Z, using the another control model.



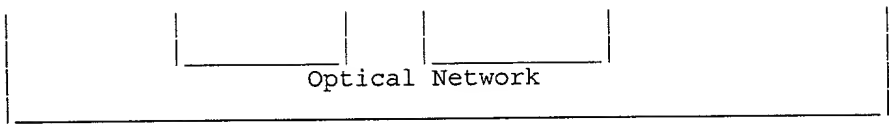


Figure 2

Description: An edge node, A, in a client network recognizes the need for additional bandwidth to another edge node, Z. Node A is both the Trail Control Requester and the Trail Control Head as shown in figure 2. As Trail Control Requester, it creates a connection setup request. As Trail Control Head, it assigns a unique Trail ID to the connection and sends the CREATE message (1) directly to edge node Z, the Trail Control Tail. It need not know the port to be used at node Z. Node Z, the Trail Control Tail, can provide this information before forwarding the request to the optical network. The Trail Control Tail forwards the CREATE message (2) to the ONC via the ONE connected to the selected port. This ONE forwards the request to the Optical Network Controller (ONC) to manage the connection across the optical network.

Once the ONC completes the connection across the optical network, the ONC sends a CREATE ACK to the Trail Control Tail, and the Trail Control Tail sends the CREATE ACK to the Trail Head, which is also the trail Requester in this case..

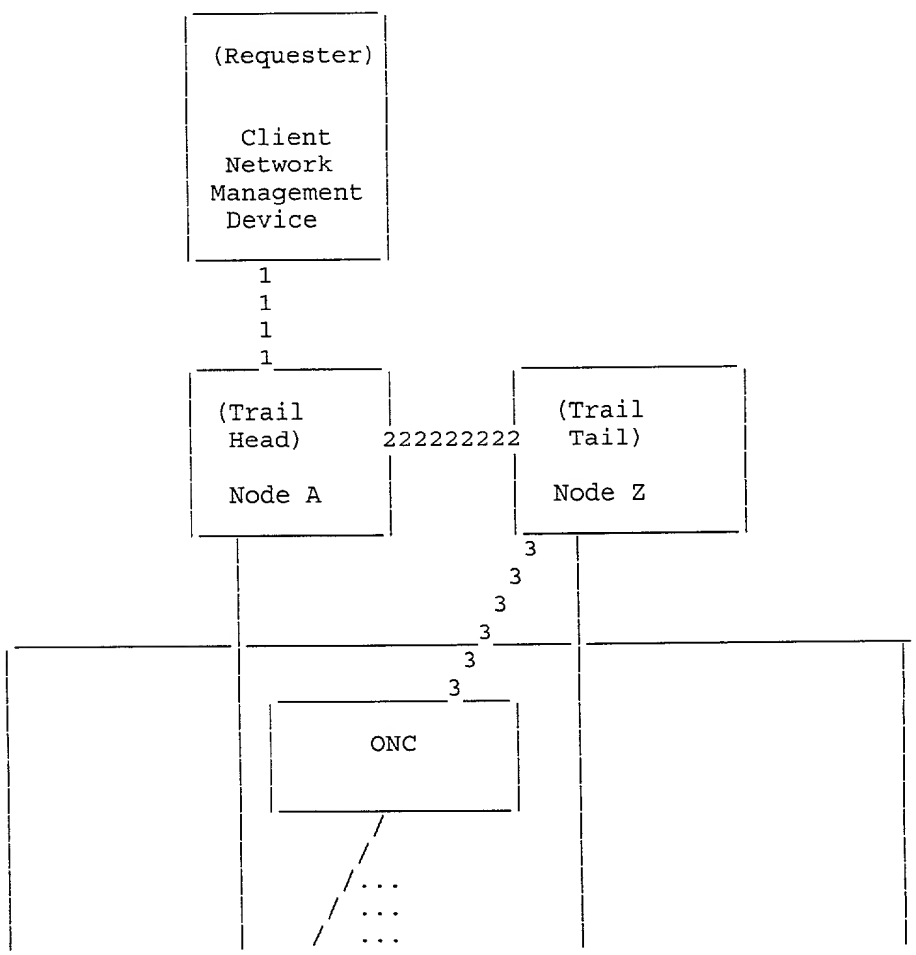
Comments: As in example 1, the edge devices in this scenario could be any two devices that cooperate using ODSI signaling control. They could be a pair of routers in an IP client network, or a pair of ATM switches, or a pair of SONET/SDH ADMs in two separate metro networks.

In this scenario, the client network can be self-managing. No management system is needed for the client network. The management intelligence can be within the edge nodes A and Z. All that is needed is the criteria to initiate a request for more bandwidth, the ability to select the ports to be used locally, and the ability to execute the ODSI protocol and control model.

This scenario can only be used after IP communication is possible between the Trail Control Head and Trail Control Tail.

Appendix B.3: Example 3

Example 3: A management device sends a request to an edge node, A, in a client network additional bandwidth to another edge node,Z, using the "new" ODSI control model.



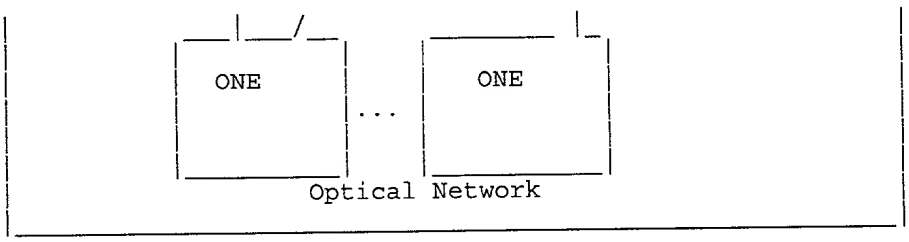


Figure 3

Description: A management device in a client network recognizes the need for additional bandwidth between edge nodes A and Z as shown in figure 3. The management device acting as the Requester creates a connection setup request that it sends (1) to the Trail Head, node A.Edge node A acting as the Trail Head assigns a unique Trail ID to the connection and sends (2) the request to edge node Z, the Trail Tail. Node A need not know the port to be used at node Z. Node Z, the Trail Tail, can provide this information before forwarding the request to the optical network. The Trail Tail forwards (3) the complete request to the ONC via the ONE connected to the selected port. This ONE forwards the request to the Optical Network Controller (ONC) to manage the connection across the optical network.

Once the ONC completes the connection across the optical network, the ONC sends a completion ack to the Trail Tail and the Trail Tail sends the completion ack to the Trail Head, which sends a completion ack to the Requester (with the Trail ID).

Comments: As in example 1, the edge devices in this scenario could be any two devices that cooperate using ODSI signaling. They could be a pair of routers in an IP client network, or a pair of ATM switches, or a pair of SONET/SDH ADMs in two separate metro networks.

This scenario requires IP communication between the Trail Head and Trail Tail. This communication may use a separate control network. It may use in-band communication on existing connections between the Trail Head and Trail Tail when they already exist.

In this scenario, the management device acting as requester may be an centralized management system in the client network or a PC terminal acting management station. Alternatively, the management device might be an intelligent client node in a client network with self-managing NEs.